UNIVERSITY CEU - SAN PABLO

POLYTECHNIC SCHOOL

BIOMEDICAL ENGINEERING DEGREE

BACHELOR THESIS

Adaptive sentence similarity applied to biomedical literature

Author: Juan Gonzalez Cabello
Supervisors: Carlos Oscar Sorzano

July 2021

UNIVERSIDAD SAN PABLO-CEU
ESCUELA POLITÉCNICA SUPERIOR
División de Ingeniería

Datos del alumno

NOMBRE:

Datos del Trabajo

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:
SECRETARIO:	FDO.:
VOCAL:	FDO.:

Reunido este tribunal el _______/_______/_______, acuerda otorgar al Trabajo Fin de Grado presentado por Don___________________________________la calificación de ___________________.

ACKNOWLEDGMENTS

Me gustaría agradecer a los profesores que han tenido especial paciencia y dedicación conmigo, aunque no siempre haya sido fácil. En especial a Gabriel Caffarena y Constantino García. En segundo lugar, al tutor Carlos Oscar Sorzano, que se ha involucrado y ha guiado este proyecto, preocupándose por que todo avance correctamente durante el año que ha durado este desarrollo. Por último, a mi madre por dejarme un lugar donde trabajar y a mis compañeros de universidad Diego Lopez y Jorge García que con su motivación han hecho posible llegar al final de esta etapa. Por último, mención especial a Gonzalo Bayón, que ha tirado de nosotros en los peores momentos y que ha puesto siempre todo lo que está en su mano por que sus compañeros salgan adelante.

ABSTRACT

Artificial intelligence has been, and still is, a key element in the development of biotechnology-related technologies, especially for disease diagnosis and drug development. One of the main advances has been in natural language processing (NLP), an area of modern computing that has been studied for decades and in which many goals are still far from being achieved.

The goal of this project is to learn a metric distance between sentences related to biomedical topics. For this we have created a database from thousands of biomedical documents and, using state-of-the-art natural language processing techniques, we reduce each sentence into a high-dimensional vector. These vectors are mapped by our model to a lower vector space in which phrases coming from the same documents have smaller distances.

The model we propose has a technical relevance that relies on the training of a neural network using triplet losses that try to reduce the distance between sentences of the same class and increase the distance between sentences of different class.

RESUMEN

La inteligencia artificial ha sido, y sigue siendo, un elemento clave en el desarrollo de las tecnologías relacionadas con la biotecnología, sobre todo para el diagnóstico de enfermedades y desarrollo de fármacos. Uno de los principales avances ha sido el del procesamiento de lenguaje natural (NLP), un área de la computación moderna que se estudia desde hace décadas y en la que todavía muchas metas están lejos de ser alcanzadas.

El objetivo de este proyecto es aprender una distancia métrica entre frases que tengan relación con tópicos biomédicos. Para esto hemos creado una base de datos a partir de miles de documentos biomédicos y, haciendo uso de las técnicas más avanzadas de procesamiento de lenguaje natural, reducimos cada frase en un vector de alta dimensión. Estos vectores son mapeados por nuestro modelo a un espacio vectorial inferior en el que tienen distancias menores las frases que vienen de los mismos documentos.

El modelo que proponemos tiene una relevancia técnica que se apoya en el entrenamiento de una red neuronal utilizando pérdidas de tripletes que tratan de reducir la distancia entre frases de la misma clase y aumentar la distancia entre frases de distinta.

INDEX

FIGURE INDEX

TABLE INDEX

1 INTRODUCTION

1.1 Objectives

Biomedical engineering has advanced by leaps and bounds in recent decades. The area of health has always been one of the most important fields of science and in the times in which we live in the main path it must take in order to evolve is through technology. This is the reason behind why the figure of the biomedical engineer is gaining more and more relevance as a support to the usual health workers (clinicians, pharmacists, researchers...). Biomedical engineering can be divided into two branches, hardware and software:

- **Hardware:** Robotics belongs less and less to the world of science fiction, the amazing advances that have been made in the last century continue to save millions of lives today. These advances range from the first hospital machines such as computer tomography or X-ray machines to exoskeletons and robotic prostheses being developed in the world's most prestigious laboratories. Although great things have been achieved, there is still a long way to go in this field, because with the arrival of new technologies such as artificial intelligence or blockchain, everything points to the fact that they can be used to help global health.

- **Software:** On the other hand, we can also find great advances in the field of algorithms and programming. Mathematics and statistics have gained unprecedented importance in recent years. With the advent of artificial intelligence, statistical models designed by engineers and mathematicians are capable of performing tasks that were previously reserved for humans. And not humans in general, but experts in the subject matter at hand. An example of this is how, thanks to the development of computer vision, a computer can analyze an X-ray and establish probabilities of finding positives of certain diseases in the patient, offering support to doctors' decisions that before could only be achieved with years of study and dedication. And where recent algorithms excel most is in tasks that humans cannot do, such as processing and drawing conclusions from millions of nitrogenous bases in DNA or reconstructing the structure of a protein that exists at the molecular level using electron microscopy images.

These branches are not independent but support each other to make more interesting and enhance each other's discoveries. A robotic arm improves considerably if it is implemented together with an object recognition algorithm since it can be used to make life easier for handicapped people [1]. Similarly, a microprocessor can be used to increase the portability of a model and can be used in emergency situations such as an algorithm for the monitoring and detection of heart disease.

Our goal in this project is to go one step further in the development of software applied to biomedicine by developing a diagnostic system that uses natural language processing to search for similarities between a patient's symptoms and the description of a disease. For this we have used models that represent sentences in a geometric space of high dimensionality, these representations in the form of numerical vectors are called embeddings. These representations are generic for language, but we have trained a second model that reduces the Euclidean distances between embeddings by establishing relationships in a purely biomedical context. The motivation is to build a classifier that has as input a sentence describing the symptoms of a patient and can predict to which document it belongs based on Euclidean distance between embeddings.

For this purpose, we have created a database with text files containing phrases related to a specific disease (one disease per document). In this way we managed to reduce the distance between phrases that belong to the same file and to separate in the vector space the phrases that belong to different documents. We have achieved this by using a penalty function called triplet loss. Using triplet loss allows us to train a model that accepts as input three sentences at a time, (two of the same class and one of a different one) and it learns to represent the sentences in such a way that the first two are together and these are separated from the last one.

1.2 Background

Among the new technologies that have been developed, we have been able to take advantage of natural language processing (NLP) to meet the objectives that had been proposed in this work. NLP is the set of techniques and technologies used to understand and represent languages that have been generated by humans in a natural way. It is a field of study that has been developing for several decades but is still far from reaching its full potential.

The strength of this methodology is the ability to represent words and phrases in a high-dimensional geometric space. For this, language elements are transformed into numerical vectors so that representations of words that have similar meanings are close together in the vector space. This allows us to train deep learning models such as neural networks that work mostly with numerical inputs.

On the other hand, when trying to classify sentences in hundreds of classes (one for each document or disease) it is not enough to use a conventional classifier, so we have been forced to use a penalty function that is able to handle a large amount of data in high dimensionality spaces. The method we have used is called triplet loss, which tries to minimize the distance between a baseline input (anchor) and one of the same class (positive), while maximizing the distance between them and an input with a different label (negative).

Since it is difficult, if not impossible, to visualize data with hundreds of dimensions, it has been necessary to transform our results with a dimensionality reduction, for which we have used t-SNE. T-SNE is a technique for data visualization that consists of reducing the dimensionality of the data while maintaining its structural integrity. The algorithm maps the probabilities of finding together all combinations of points in the high dimensionality space and tries to create a set of points with a similar distribution but with lower dimensionality, maintaining the clusters of the original dataset.

1.3 Critical Analysis

So far, new technologies have been used for numerous tasks in the field of biotechnology, whether to handle immense amounts of patient data, support specialists in diagnosing diseases, robotics to supplement the needs of people with disabilities, or even machines that can operate remotely controlled by a surgeon. Deep Learning alone has already achieved accomplishments that seemed impossible decades ago, such as surpassing in many cases human capabilities in certain tasks.

Despite this, natural language processing still has some way to go. So far it has been used for generic tasks using models that are not specialized for medical data such as chatbots or assistants for doctors. Where NLP is most useful is when processing the reports that a doctor writes about a patient. Key information that specialists write is lost in these text reports but is difficult for a computer to extract in a way that can be properly analyzed. These huge volumes of unstructured data are processed with engines capable of scrubbing large sets of unstructured health data to discover previously missed or improperly coded patient conditions.

The problem is that medical language is not as simple as the language used on a daily basis. Biomedical texts are full of technical terms and acronyms that can confuse even the most sophisticated model. Thus, NLP models are used that are not trained to deal with this kind of complex data and do not understand the medical context as well as they should. For this reason, several articles have started to emerge dealing with the issue of biomedical language processing, since many statistical models for processing text perform extremely poorly under domain shift.

An example of this are the models for the Disambiguation of acronyms and words that can have more than one possible meaning, which in biomedical texts become especially recurrent [2]. This is an improvement, but falls short of being a NLP model as such that can perform well in biomedical texts. To achieve this, numerous articles have been written on the subject, but the vast majority propose something as simple as fine-tuning existing models or basic transfer learning techniques.

The main alternative to what we propose is BioScentVec by chen et al., a sentence embedding model for biomedical text data [3]. This proposal outperforms the state-of-the-art as it re-trains an encoder using health-related data, but does not bring any advances to the way these models are built. Ours is not based on the training of another model but takes as input the output of other encoders and reduces its dimensionality by forming clusters of sentences in the vector space grouping sentences that talk about the same disease. This has several advantages such as not having to retrain an encoder from scratch, which saves computational and time costs since these models usually have millions of weights to modify in each training iteration. Another advantage is that different encoders can be used to generate the inputs of our model without the need to modify it, which allows us to compare different implementations easily looking for the latest results. Finally, our model, by using an unconventional cost function such as the triplet loss, allows us to form a classifier simply by measuring the Euclidean distance between the embedding of the sentence to be classified and the rest of the sentences already labelled in our database, assigning it the class with the highest similarity.

1.4 Book Structure

The remaining chapters of this paper contain the following information:

Chapters 2, 3 and 4 covers the background of the project. These chapters cover the state-state-of-the-art of natural language processing and distance learning methods so when discussing about the architecture and functions used to train the model there is a prior understanding of the mathematical concepts that support the ideas implemented.

Chapter 5 exposes the entire process that has been followed to achieve the goals proposed for the project as well as the materials used in each step. It extensively covers the details of each tool and technique used and explains the code and algorithms that make up the project.

Chapter 6 presents the results obtained and compares them to other sentence embedding models.

These results and their implications are discussed in Chapter 7. Chapter 8 is a conclusion and also sets out the future directions that the project could take to improve the results obtained.

2 Natural language proccesing

During the last decades, language has been one of the main research objects in the area of computational techniques. An attempt has been made to analyse the text based on theories and technologies that have been developed over the years and grew the furthest with the rise of computers for the purpose of performing useful tasks. These techniques for automatically manipulating natural language have been called Natural Language Processing (NLP). From a scientific perspective, NLP aims to model the cognitive mechanisms underlying the understanding and production of human languages. From an engineering perspective, NLP is concerned with how to develop novel practical applications to facilitate the interactions between computers and human language [4].

Before addressing the different methods used for text processing, it is necessary to understand what natural language is and why its study is a challenge, which despite having been a priority for years, continues to be developed and is still far from over.

2.1 Natural language

Natural language refers to any system of communication used by humans that has evolved naturally through use of the language and repetition without being constructed artificially. We avoid including animal communication in our definition of language as it is controversial and does not add value to our study.

The main form of natural language we are interested in using as data is mainly text and speech. If you think about how many times a person reads text during the day, the examples are unlimited: signs, books, user guides, etc... And there is no need to talk about the importance of the use of speech in people's lives, every interaction between two humans is done the vast majority of the time through words, even more than through text.

2.2 Challenges of natural language as data

Language is a very hard thing to learn. Just as it is complicated for a child who spends the first years of his life learning to speak, or it is complicated for an adult person trying to learn a new language; it is also hard for the engineer who attempts to build systems that deal with natural language input or output. [5]

Natural language is complicated mainly because it is messy. It has synonyms, ambiguities, different expressions to refer to the same entity, and, above all, it has few rules. Despite this, we understand each other most of the time and can infer the context from a sentence we hear, and not only this, but also the speaker's emotions, intention, personality and even style.

A widely used example of the complexity of language processing is sarcasm, it is very complicated to tell a machine that it must interpret an expression that is used to say the opposite of what is true.

Las principales tareas que pretenden resolver los sistemas NLP son:

- **Word Sequence Tasks**: Solve problems related to the sequence of words and the order they follow. Used for text generation eq. Chatbots or translators.

- **Text Meaning Tasks**: The aim is to assign a meaning to the words or phrases. For this purpose, the input data are transformed into distributional vectors that collect the features of each phrase or word. This is a word representation that allows words with similar meaning to have a similar representation. Used for sentence embedding or finding similar words.

- **Text Classification Tasks**: Predicting tags, categories or sentiment. Used for filtering emails or classifying documents. In our project we solve this task using techniques meant to solve text meaning tasks: We represent text in a way that sentences from the same document

In our project we solve classification tasks using techniques meant to solve text meaning tasks: We use a standard Natural Language Processing (NLP) embedder that transforms input sentences into high-dimensional vectors. Then, we will map these vectors into a lower dimensional space in which two embedded vectors should have a smaller distance if they come from the same document.

This sentence metric will allow in the future to identify the relevant topics related to a description of a clinical history provided by a person.

2.3 Natural Language Processing

Liddy defines Natural Language Processing as a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications. [6]

It is important to emphasize the concept of the different levels of linguistics that she proposes. These are seven, with their respective forms of processing:

- **Phonology**: Interpretation of speech sounds within and across words. When developing a natural language processing system that accepts sounds or spoken speech as information to be processed, it is necessary to encode the signal into a digital format so that it can be interpreted by the model used.

- **Morphology**: Componential analysis of words. Each morpheme in the word is analysed including prefixes, suffixes and roots. An NLP System can recognize and assign importance and meaning to each morpheme to obtain more detailed information about each word.

- **Lexical**: Word level analysis including lexical meaning and part of speech analysis. Each word is analysed individually by the NLP System, but different types of processing are applied for the meaning assignment. In this level, words are considered just part of speech, assigning the most probable part-of-speech in case of ambiguity.

- **Syntactic**: Analysis of words in a sentence in order to uncover the grammatical structure of the sentence. Words maintain relationships of structural dependence between them, this is the information obtained from this level of processing.

- **Semantic**: Determining the possible meanings of a sentence, including disambiguation of words in context. This might seem to be the only level of processing that contributes to the assignment of meaning, but as we have already seen, it is the set of all the previous levels that make the correct determination of meaning. An important part of this level is the disambiguation of words, a single possible meaning is selected from polysemous words.

- **Discourse**: Interpreting structure and meaning conveyed by complete texts beyond a single sentence. NLP systems should take text as a whole, focusing in dealing with the structure of the text or connections between sentences.

- **Pragmatic:** understanding the purposeful use of language in certain situations. NLP systems deal with the use of real-world knowledge and understanding of how this impacts the meaning of what is being communicated.

In any case, the lower levels are the most relevant in the study of natural language, since it is often not necessary to reach higher levels depending on the application of the system. On the other hand, it is easier to advance in the development of the lower levels of processing because these are based on dealing with small units of language such as phonemes, morphemes or words, guided by rules, which are easier to analyse than complete texts, contexts or emotions.

For our project we use models that work optimally with complete complex sentences, so we deal with the first five levels of language processing, leaving aside the discourse level because we do not get to analyse complete texts, more than a couple of diagnostic sentences, and we do not deal much with the pragmatic level because the diagnoses and symptoms of a patient do not usually contain a purpose that depends on the situation or the context.

In the fragment of Lilly's definition where she mentions computational techniques, she refers to the different approaches that can be taken when processing natural language. Li Deng divides these approaches of the general methodology used to study NLP, from a historical perspective, into three waves listed in the following sections:

- **Rule-Based Approach**: (First wave) These are handcrafted system of rules based on linguistic structures. The experts designed these programs using symbolic logical rules based on careful representations and engineering of such knowledge [4]. Rule-Based systems have been proven to work well specifically but fails when generalizing. The biggest throwback is that building these kinds of models require expert knowledge.

- **Machine Learning or 'Traditional' Approach:** (Second wave) Started with the implementation of shallow machine learning and statistical models with the explosion of data corpora. Comprises probabilistic modelling likelihood maximization, and linear classifiers. The system analyses data from an annotated training set and develops its own classifier. This still needs to be complemented with hand crafted features needing some type of expertise. In this second wave, shallow generative models such as hidden Markov models HMMs [7] began to be used for spoken language understanding and speech recognition. Later on, Deng and some colleagues made use of multiple latent layers of representations for this generative process giving rise to the first deep learning industrial process [4].

- **Neural Network and Deep Learning Approach:** (Third wave) Shallow machine learning from earlier days did not have the capacity to absorb large amounts of training data, so the algorithms were not powerful enough. This changed a few years ago with the popularization of deep learning which achieved best results compared to traditional machine learning algorithms. The reason of this progress was mainly that in traditional machine learning, human experts were needed for feature engineering. Deep learning, by using deep and layered model structures, broke the difficulties related to the need of manually crafted features. These models are capable of extracting features by using a cascade of layers of nonlinear processing units, learning a representation of the language from scratch and solving general machine learning tasks dispensing with feature engineering. The biggest impact of these models has been in the field of speech recognition. It hardly affects our project apart from the fact that the main industrial implementations (Microsoft Cortana, Apple Siri, Amazon Alexa, Google Assistant) are based on Deep learning models, but it is a good example given the huge impact it has had on technology in recent years. Despite this, the aforementioned systems are still used for certain applications. These are the models explored in this project. [Deep Learning in Natural Language Processing, 2018]

3 Word Representation

[Huang et al, 2019] [Chollet, 2018] Like all other neural networks, deep-learning models don’t take as input raw text: they only work with numeric vectors. Vectorizing text is the process of transforming text into numeric vectors or tensors. On the other hand, NLP have the issue of representing the features from the original text data, so the goal is that these vectors store semantic information, which allows them to be associated or dissociated to other vectors (words) according to different grammatical contexts.

Word-vectorization starts dividing or breaking words in different units called tokens by a process called tokenization. Once the words have been tokenized, vectors of numbers are associated to each word, obtaining the numerical representation of the vectors. These number vectors are used as input for the natural language processing neural network. which is normally applied with words as input but in the case of our project a model for encoding whole sentences is used, the universal sentence encoder.

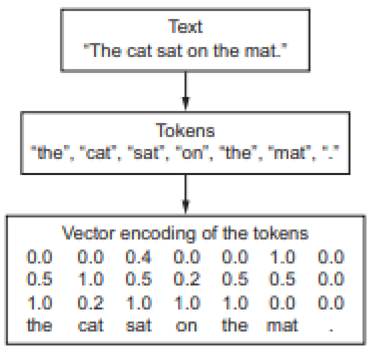


Figure 1. Tokenization of words

3.1 One-Hot Encoding

One-hot encoding is the most common, most basic way to turn a token into a vector. In this method for vectorization of words each word is associated with a unique integer index i in order to obtain a vector of length N (the size of the vocabulary) whose components are all zeros except the one corresponding to position i, which will have value 1. The result is a vector whose elements are only 1 and 0. Each word is written or encoded as one hot vector, with each one hot vector being unique. This way the word can be identified uniquely by its one hot vector and vice versa, so no two words will have same one hot vector representation.

To clarify, let's say that the word "calculator" occupies position 300 in the list of words in the alphabetically ordered vocabulary, and that the vocabulary is made up of 171,476 words. In this case the vector associated to the word "calculator" will be a vector of length 171,476 in which all its positions will have a zero value except for position 300 which will have a 1. The same can be applied for the rest of the words of the alphabet or for characters being the vectors of the latter of length 26 (number of letters in the alphabet).

This way words are represented by vectors and all words in a sentence can be represented by a matrix constituted by those vectors. The same way a text made up of different sentences can be represented by an array of matrixes, a three-dimensional vector that the neural network will accept as input.

There are two major issues with this approach for word embedding. The first one is related with dimensionality, even if you do not use the complete alphabet dimension for the encoding, one-hot vectors will always be very high-dimensional and therefore require large memory space. One-hot encoded data is sparse since the vast majority of the matrix is filled with zeros so, if our vocabulary has one million words, each word is represented by 999,999 zeros and a one. The fact that memory consumption is exponential makes it a very computationally inefficient vectorization method.

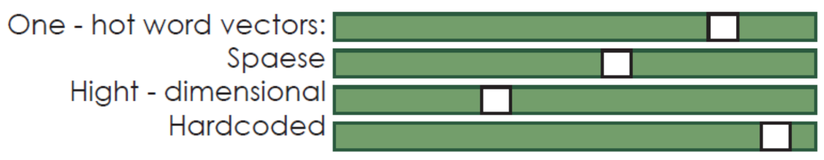


Figure 2. One-hot encoding vectors

The second issue is that there is no meaning associated to the numeric vector. Because the representation is N-1 zeros and a one, being N the number of dimensions, the resulting vector do not say much about about the words they represent. For example the words "dog", "cat" and "squirrel" have a similarity between them that can be easily seen since they are animal species, so it would be logical to classify them together. With One-hot vectors this does not happen because you cannot extract any information or features from the vectors so all the words are at the same distance from each other.

3.2 Word embeddings

Word embeddings are dense vector-representations of words. This method of vectorization maps each word to a N-Dimensional space being an interesting and powerful way of associating vectors with words. These real valued vectors are not arbitrary generated, they are learned through supervised techniques such as neural network models trained on tasks such as sentiment analysis and document classification or through unsupervised techniques such as statistical analysis of documents.

Word embedings:
Dense
Lower - dimensional
Leamed from data

Figure 3. Word Embedding Vectors

Complex vectorization methodologies, such as word embedding, emerge from the need to solve three major problems that arise when using simpler forms of vectorization:

- **Scalability**: Simplistic methods such as One-Hot can reach millions of dimensions if the entire vocabulary of a language is considered. The computational resources and time required to train a neural network by feeding it data with such a large number of dimensions is at least inefficient.

- **Sparsity:** Is really hard to train models if your data is made up by millions of zeros and some ones in the right locations. This prevents the model from generalizing correctly when dealing with test data.

- **Context:** Creating vectors without taking into account the context and dependencies between words, leads to a lose of semantic information. This is really problematic because of similar words having totally different representations.

Contrary to the vectors obtained after applying one-hot encoding, word embeddings are low dimensional, it’s common to see word embeddings that are 256-dimensional, 512-dimensional, or 1,024-dimensional floating-point vectors in contrast to a million-Dimensional space. Since each vector in the embeddings is densely populated as opposed to sparse vectors made up almost entirely of zeros, this methodology also resolves the sparsity issues. Word embeddings learn from data in a way that captures the shared context and dependencies among the words, one-hot encoding generates vectors without considering the context in which each word of vocabulary lies, meaning that not only more dimensions are used, but also less information is captured.

There are many ways to represent a word with a vector, one possibility being to simply choose a random vector. The problem with this is that the embedding space would have no structure. This vector would not provide any improvement over the one-hot vectors because it would be completely arbitrary and two synonymous words could end up with completely different vector representations even though the meaning should have similar embeddings.

Word embeddings are meant to map human language into a geometric space [chollet]. Vectors have geometric relationships: they can be close to each other in geometric space, far apart, they can be in the same direction with respect to another point, etc... The purpose of word embeddings is that these geometric relationships are in accordance with the semantic relationships of the words they represent. This means that if a word is synonymous with another word, both are related by their meaning, therefore, their embeddings should be similar and close in the geometric space. Likewise, words that have different meanings will be vectorized in representations that are far from each other. In addition to distance, specific directions in the embedding space should also be meaningful.

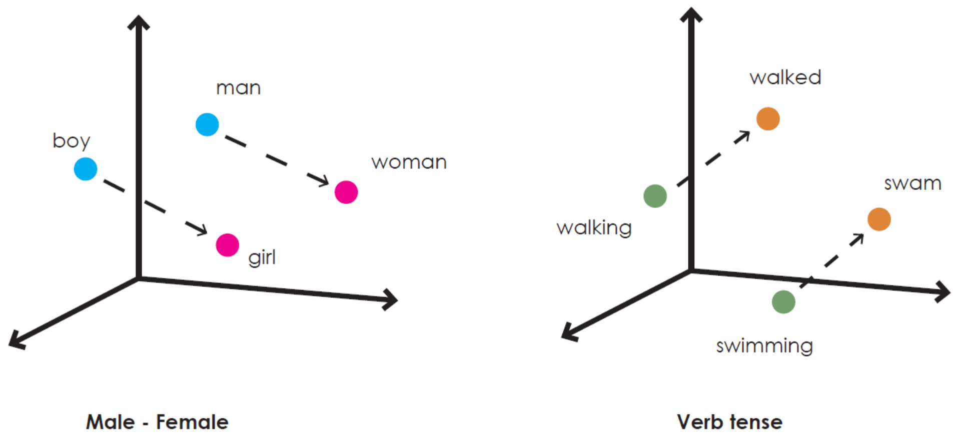


Figure 4. Distance and relations between embedded words

We take the figure 4 above as an example. In the figure, four words are embedded in a three-dimensional plane: boy, girl, woman and man. It can be seen that the words have geometric relationships according to their semantic connection. The horizontal axis could be interpreted as a vector representing gender (from male to female) while the vertical axis would be a geometric transformation from early age to adulthood (from boy to man). The same can be applied to other types of words like verbs, being the verb tense one possible transformation.

The ideal word-embedding space would be one that maps the English (or any other) language perfectly representing the relations of every word on the dictionary. It is yet to be computed but it could never be perfect because each language depend on specific cultures and the context and relation between words varies among different tasks. The interpretation required to read a legal document is not the same as that required for a cooking recipe, the importance of each word changes depending on the task to be performed. In spite of this, we can compute embedding spaces appropriate to the task at hand. We can visualize these spaces using TSNE visualization (Explained in other title) and would look like figure 5:

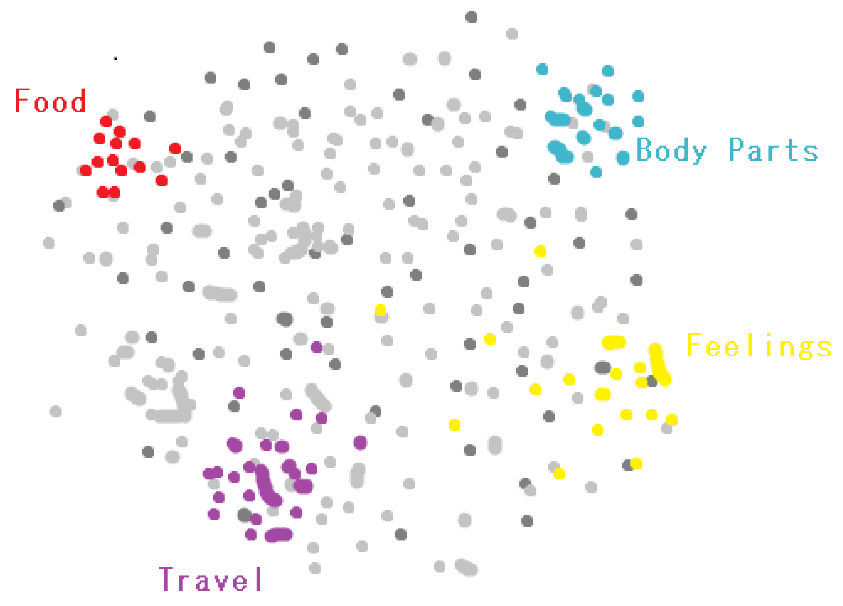


Figure 5. Scatter plot of word embeddings

See how names of food, colored in red, are al together on the top-left of the graph, while feelings and body parts are in an opposite direction but close to each other, because those families of words have some kind of semantic relation.

Although we do not use word embedding as such in our project, it is appropriate to review the main algorithms that led this vectorization methodology to become the state-of-the-art in word representation:

3.2.1 Word2Vec

This algorithm was proposed for the first time by Thomas Mikolov et al. in 2013 with the paper "Efficient Estimation of Word Representations in Vector Space". It takes advantage of the idea of distributional hypothesis, which suggests that words that we can find used in the same context also have semantic relationship. Knowing which words have the same semantic meaning, a mapping of words can be made so that they are close to each other in the geometric space.

Word2Vec consists of training 2-layer neural networks, the first layer being a simple model that learns continuous word vectors and then trains N-gram NNLM on top of these representations (NNLM was the state-of-the-art neural network model by the time that word2vec was proposed). [8]. This 2-layer model takes as input a large set of sentences (corpus) of text and produces a vector space with hundreds of dimensions. Each word in the corpus is assigned to a single vector in that space [9].

There are two different flavours of this algorithm, continuous bag of words (CBOW) and skip-gram:

- **CBOW:** Given a corpus the model loops on the words of each sentence and tries to predict the conditional probability of the current word given its context.

- **Skip-gram:** Skip-gram is a flipped version of CBOW. The goal is to predict the surrounding context words given a single target word. A feature vector is used as an input to the model, when the network fails to predict the surrounding context, the components of the vector are adjusted.

This algorithm was a major breakthrough in the field of Word Embeddings because rather than training the model against input words, it trains words against other words that neighbour them in the input corpus [10]. This way the relations that the model captures in the algebraic representation are in a way that they have never been captured before. For example, if we map the words “boy”, “girl”, “man” and “woman” in to the vector space, we found out that the word “man” is the similar to “boy” in the same sense that “woman” is similar to “girl”. This way we can find a word that is similar to “girl” in the same sense as “man” is similar to “boy” by computing the vector [11]:

```
Wrd2Vec[“Man”] - Wrd2Vec[“Boy”] + Wrd2Vec[“Girl”] =
             Wrd2Vec[“Woman”] (1)
```

3.2.2 Glove

GloVe [12] (Global Vectors for Word Representation) arise after a brief silence in the world of word representation as an improvement to Word2Vec. The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context surrounding words) to derive semantics of a word but combines them with global statistics (word co-occurrence) to obtain word vectors and semantic relationships between them [13].

Without going to go into detail, we review this embedding because as it is a precursor of some of the models we use in the project. GloVe method is built on the idea that using a co-occurrence matrix, semantic relationships among words can be derived. This is a matrix containing information on how many times a word co-occurs next to another in the same context.

Whit this data we can compute the ratio of a word occurring against a pair of words. Let's say we have three words i, j and k: this algorithm compute the probability of finding words i and k together (P_ik) and the probability of finding j and k together (P_jk). With the ratio P_ik/P_jk the algorithm obtains global information about the relative position of word k with respect to words i and j in the geometric space. GloVe uses this mapping of words, added to the Word2Vec local statistics come up with a principled loss function that is used to train the model that generates the embeddings. This incorporating global statistics to the model. This is an improvement because leverages both global and local statistics of a corpus.

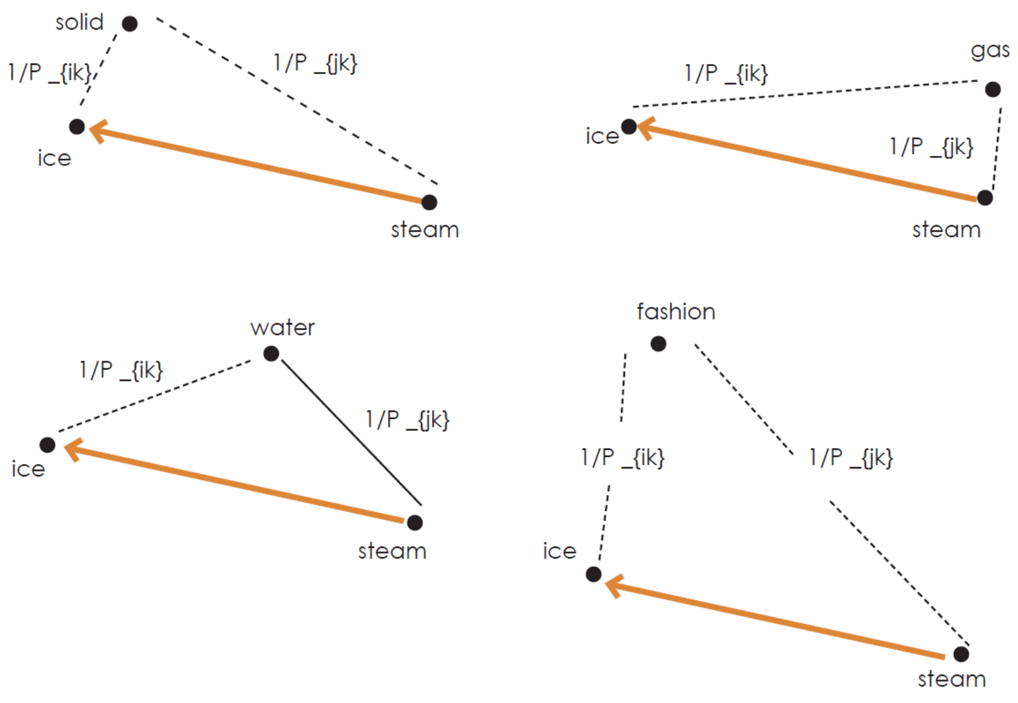


Figure 6. GloVe ratio of words representation

3.2.3 ELMo

[14] Word embedding models such as word2vec or Glove were a great improvement for NLP since they could be used as pre-trained models. These models had been trained with huge amounts of data and by using them as a starting point to build models dedicated to specific tasks, it was possible to obtain good results without having enough data to train. Comparing a sentence classification model used from scratch with one that uses a pre-trained model gives different results. The former will not be able to handle or understand the words that do not appear in the training and being built from scratch it has to learn the embedding, which consumes more training time. The second one, on the other hand, can handle many more words and since it is an almost perfect model from the beginning of the training, it will only have to be adapted to the task to be performed.

Although these models were a breakthrough for the NLP situation at the time, these models were not perfect. Because of this, people studying NLP began to raise a problem that had been overlooked. This was that a word with several meanings would get only one representation in vector space when vectorized by these encoders. It had been possible to reduce the distance between vectors representing different words that have the same meaning but not the opposite. Take for example the word "organ", sometimes it means a part of an organism, but other times it refers to a musical instrument depending on the context. For a pretrained Word embedding it is impossible to assign different representations in the vector space depending on the context. Therefore ELMo was developed to solve this problem.

heart
organ
organ
piano

Figure 7. Representation of polysemic word

ELMo is a function that, instead of using a fixed embedding for each word, it looks at the entire sentence before assigning each word in it an embedding. So ELMo could represent the word organ close to words related to body parts when the sentence was "The heart is a vital organ". On the other hand it could output another different embedding if the input sentence was "I like to play the organ on Sundays" so that in the embedding space it would be placed near other musical instruments. The same happens with other types of words such as verbs, which depending on the pronunciation mean one tense or another.

To understand how ELMo works it is necessary to first address an architecture called Long-Short Term memory layers (LTSM). [15] LTSM is a recurrent neural network (RNN) architecture commonly used to process entire sequences of data in contrast to networks that process isolated data points. The main idea is that these networks can maintain information throughout training. This means that when processing a sentence, it will not do so independently for each word, but will process the first word and use what it has learned to process the next word, and so on until the sequence is completed. In this way, when the last word is reached, it will be processed taking into account the information of all the previous words. In the case of a sentence, it could be said that it takes into account the context of previous words.

ELMo consists of several layers. The first one consists of an encoder that simply depends on the syntax of the word and not on the context. This was decided by its creators so that the comparison with the pre-trained models is fair, being these and ELMo two models trained from scratch. The rest of the layers are bi-directional LTSM layers. Using this architecture, the network tries to predict the word taking into account not only the previous words but also the next one. This results in two embeddings that when added together produce the final embedding. To obtain these two vectors, ELMo collapses the outputs of all layers in the network into single vectors. It could use only the top layer outputs, but Peters et al. demonstrated that the weighted sum of all the layers give better results.

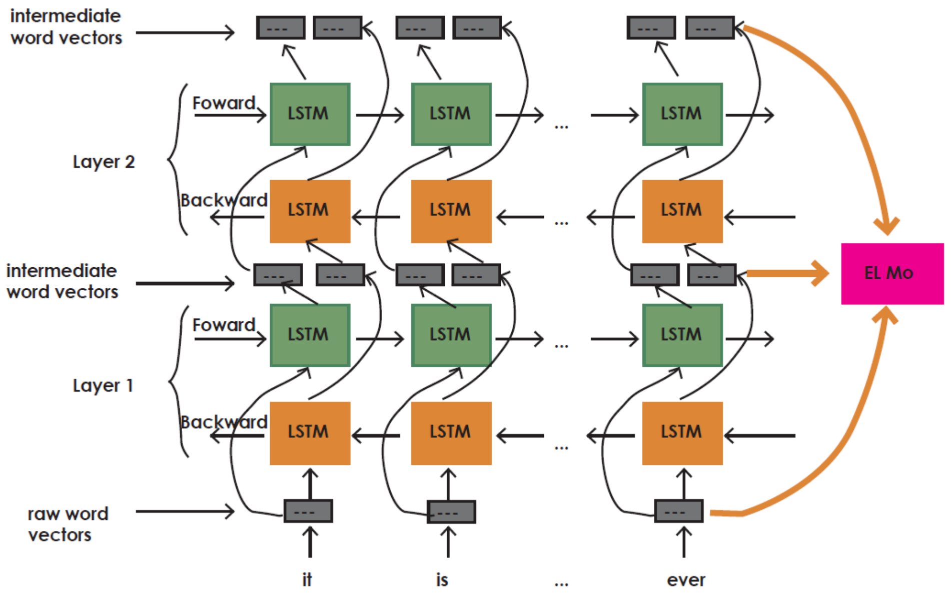


Figure 8. ELMo Arquitecture

3.3 Sentence Embedding

In recent years, the study of natural language processing has gone one step further with the development of sentence embedding. With the proposal of bidirectional networks, which processed knowledge of strings of words, researchers in this field began to consider the possibility of working with complete sentences instead of individual words. In the case of large text, using only words would be very tedious and we would be limited by the information we can extract from the word embeddings.

Sentence embedding techniques represent entire sentences and their semantic information as vectors. These vectors act as word embeddings, containing the features and information of the sentence and representing them in the geometric space in the same way it is done with words. In this section we will review different sentence embedding techniques including the state-of-the-art and the model we have used.

To represent sentences, we can't one-hot encode them as it is done with words because there are so many possible sentences that it would be unpractical. One practical way to create a sentence representation is to take advantage of the embedding of each word and calculate the embedding of the whole sentence based on those [16] (e.g. calculate an average weighted average of the embeddings of the words that form the phrase).

3.3.1 Doc2Vec

Doc2Vec [17] is an unsupervised algorithm and adds on to the Word2Vec model by introducing another 'paragraph vector'. It creates representations of a document, regardless of its length. But unlike words, documents do not come in logical structures, so you can't use the same method as word2vec.

The difference that was introduced was a new vector, the paragraph vector. Thus, when training the model with texts, each text had the vectors of each word plus a paragraph vector. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document. [17] It is a simple method but one of the first to be implemented for embedding phrases.

3.3.2 Transformers

Transformers [18] are processing units with Encoder-Decoder structure, it was developed for machine translation, but it has ended up being used for many other tasks. It is constructed by a stack of encoders that output the data to a group of encoders of the same number. All of these units are identical in structure. The encoding of the phrase is performed in the bottom encoder, and this outputs the result to the one directly above.

The key property of the Transformer is that the word in each position flows through its own path in the encoder, so each path is dependent on each other. Each encoder consists of a self-attention layer [18] and a feed-forward network. The self-attention layer associate words in a sentence. For example, in the sentence “The animal was scared because it was lonely”, this layer allows to associate the word “it” with “animal.

The encoder [19] takes each word in the input sentence, process it to an intermediate representation and compares it with all the other words in the input sentence. The result of those comparisons is an attention score that evaluates the contribution of each word in the sentence to the key word. The attention scores are then used as weights for the generation of new embeddings by the feed-forward network.

The decoder takes this intermediate embedding and tries to reconstruct the original sentence or similar using the hidden states of the encoder. Unlike the encoder, the decoder uses an addition to the Multi-head attention that is called masking. This operation is intended to prevent exposing posterior information from the decoder.

These models are mostly used in machine translation. This is because you can enter a phrase as input, have it transformed into an intermediate embedding by the encoder and then have it reconstructed by the decoder in another language.

3.3.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) [20] is a model that takes advantage of the bidirectionality structure used by ELMo but instead of LTSM layers, it takes advantage of OpenAi transformers. These consist of decoders from the transformer.

When this idea was proposed there was a problem, and that is that ELMo uses bidirectional LTSMs, but the transformers train a forward model. If the transformers had all the information of the present and future of the sentence, the correct answer would be revealed and would disrupt the learning procedure. [21] To solve this, BERT's modelling task masks randomly 15% of words in the input and asks the model to predict the missing word. Instead of predicting the next word in a sentence, it masks some percentage of the input tokens at random and predicts those masked tokens.

Another problem is that to train an embedder, you need to establish relationships between what you want to vectorize, in the case of words you use the context in a sentence to establish relationships, but in the case of phrases you have to be more precise. The relation between two sentences is stated as follows: "Given two sentences (A and B), is B likely to be the sentence that follows A?", this task forces the model to learn the relationship between two sentences, which is not directly captured by language modeling. It also establishes more common relationships to other sentence embedders such as if one sentence is a paraphrased version of another.

3.3.4 Sentence-BERT

Sentence-BERT (SBERT) [23] is a model for sentence-pair regression task like textual similarity built by modifying the original BERT network. This model is currently the state-of-the-art in sentence embedding. This approach is able to derive semantically meaningful sentence embeddings using Siamese networks (explained in chapter 4). It tries to cluster similar sentences using these kinds of networks. Then when trying to find similar sentences it uses a similarity measure like cosine similarity or Manhatten / Euclidean distance so the most similar network can be found.

This model is what inspired our project but using a different architecture. Instead of using Siamese networks we have built a triplet architecture using the triplet loss function.

3.3.5 Universal Sentence Encoder

Inspired by the use of models such as word2vec or GloVe, Universal Sentence Encoder (USE) [22] is a model developed by tensorflow to be used as a pre-trained model for different NLP tasks.

It has two variants, the first, unlike BERT, uses Transformer encoders. The architecture consists of 6 stacked transformer layers. Each layer has a self-attention module followed by a feed-forward network. Using the properties of encoders, such as self-attention layers, USE is able to extract the context of the sentence by generating the embedding of each word. The embeddings of each word are summed element-wise and divided by the square root of the sentence length, resulting in a 512-dimensional vector.

The second variant uses Deep Averaging Networks (DAN), an architecture proposed by Iyyer et al. For this, the model averages the embeddings of the words and feeds them into a 4-layer feed-forward deep neural network that outputs a 512-dimensional vector. The embeddings and layer weights are learned during training. This model has less accuracy than the Transformers version but is less computationally demanding.

For this project we have chosen USE as a pre-trained model for two reasons. First, because, being tensorFlow it is very easy to implement in Python and it is prepared for transfer learning tasks. The second reason is that there is a multilingual variant of this model that has been trained with foreign languages. This is convenient for our interests since our data is in Spanish.

4 Loss Functions / Arquitectures

4.1 Distance learning vs classificator

It is important for the project to understand the difference between a model that is trained as a classifier and a model that learns to reduce distances between elements. A classifier will modify its weights depending on whether it gets a sentence right or wrong, for example, a neural network to predict whether the input image is a dog or a cat, the input will be an image and the output will be binary indicating whether it is a dog or a cat. If it fails, the weights will be modified to increase the accuracy of the classifier.

A model that learns to reduce the distances between elements can have different architectures, but normally it compares two or more inputs and as output it indicates how similar they are. The model will learn to reduce the distance between them if they are of the same class or maximize this distance if they are different. Some of these architectures are explained in the following section.

In the case of our model the input is three vectors and the output is a modification of these vectors so that those of the same class are close to each other and those of a different class are far from each other.

4.2 Siamese Network

The main problems of a classification model are several:

- There may be a lack of data for some classes, which makes training very difficult when the model requires many examples of each class.
- If there are too many classes, the model will not be able to learn them all if it does not have lots and lots of data to learn them from.
- In a task that requires changing the data, a classifier would have to be trained from scratch each time a new class is to be introduced. This is very time-consuming and unfeasible if there are frequent modifications.

Siamese neural networks [24] are a pair of identical networks to which two data are passed to extract their features. A loss function is then used to calculate the difference between these two vectors and measure their similarity. The most common loss function used for this type of architectures is contrastive loss, which calculates the cosine distances between two embeddings. In this way this network can be used to know if two elements belong to the same class.

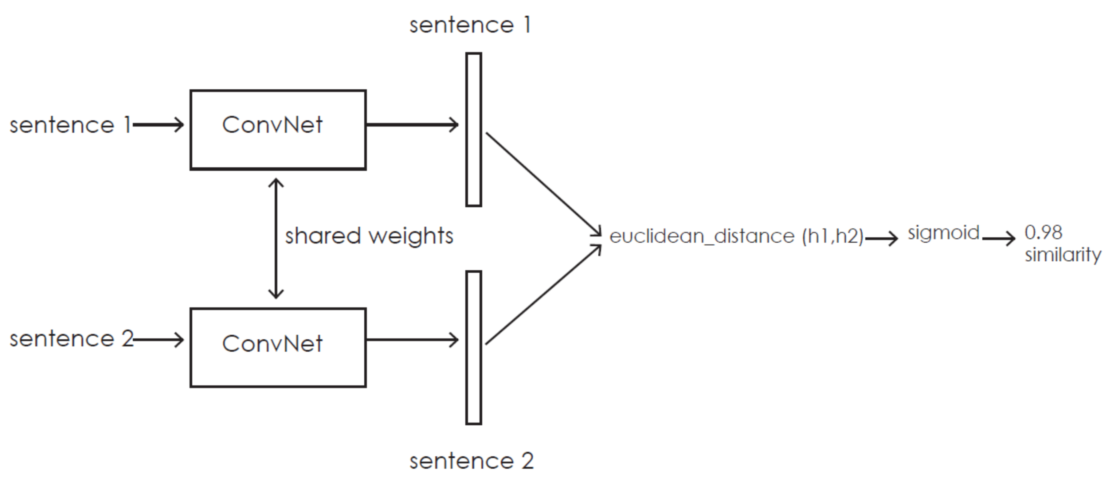


Figure 9. Siamese network arquitecture

4.3 Triplet Loss

It is a loss function that tries to optimize in two ways, that two embeddings with the same label are close to each other in the vector space, and that two examples of different class are far from each other. For this, a neural network architecture is used that takes three inputs, but these must meet specific conditions, two of them are of the same class and the third of a class different from the first two. In this way the triplet loss function will have the objective that the two positive examples are close to each other and the negative one is far away.

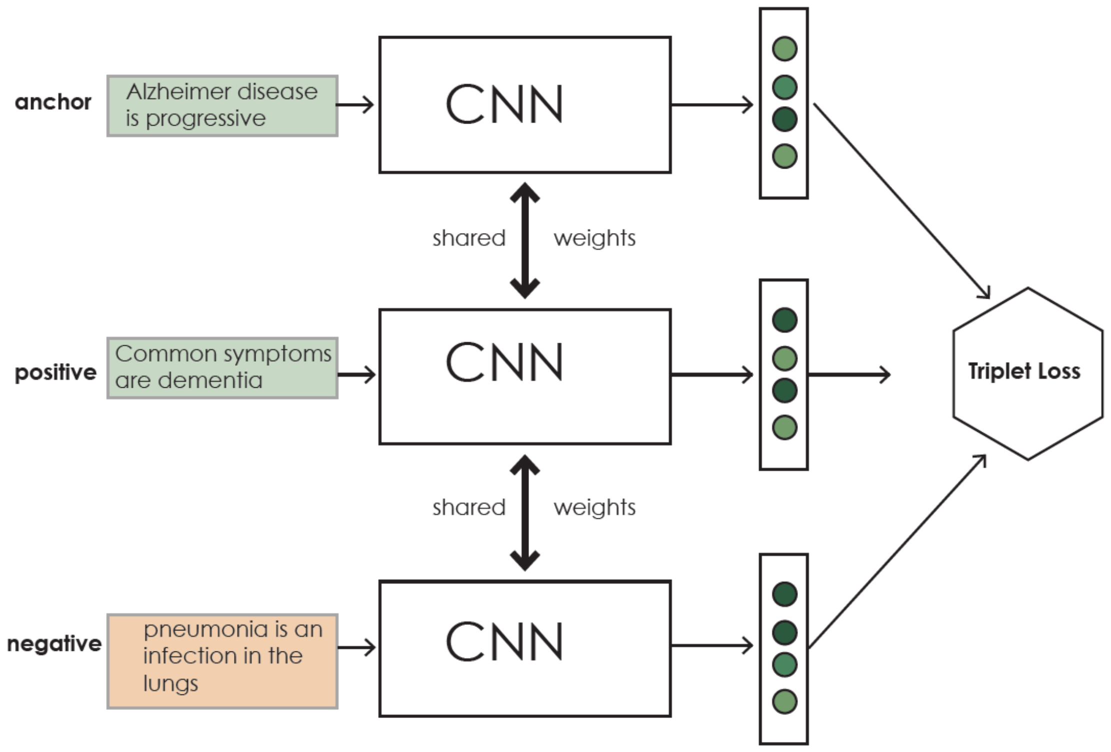

anchor
Alzheimer disease is progressive
CNN
shared weights
positive
Common symptoms are dementia
CNN
shared weights
negative
pneumonia is an infection in the lungs
CNN
Triplet Loss

Figure 10. Triplet loss arquitecture

The first input, used as a reference, is called anchor, while the one of the same label and the one of a different class are called positive and negative respectively. These three inputs form a triplet. In addition, the function uses a fourth parameter called margin. This is a forced distance between the positive and negative, as we will see now in the formula, this is necessary because otherwise the loss would be 0 whenever the negative is farther away from the anchor than the positive, even if it is by a small distance.

For a distance d in the embedding space, the loss of a triplet is defined as:

$$\mathcal{L} = \max(d(a,p) - d(a,n) + \alpha, 0) \qquad (2)$$

d(a,p) is the distance between the anchor and the positive, and d(a,n) the distance between the anchor and the negative. This distance is calculated with Euclidean distance. The difference between these allows minimizing the distance of the former while increasing the distance between the latter. This difference is compared with 0 and the maximum between the two is chosen, so that if the negative is farther from the anchor than the positive, the result of the difference will be a negative value and therefore the los will be 0. This brings the problem that, if the negative is farther than the positive, but by very little, the function will not push it farther although it would be optimal. For this reason, a margin α is added to the difference that forces the negative to exceed the distance of the positive by more than a value α.

Depending on the loss, triplets can be classified into three types, easy triplets, hard triplets and semi-hard triplets:

- **Easy triplets:** triplets that have a loss of 0, this occurs when the negative has a distance with the anchor greater than the distance of the positive with the anchor plus the margin. These are not modified because the function considers that if the negative is far enough away (d(a,p) + α) it will not be penalized.

- **Hard triplets:** These are the triplets in which the negative is closer to the anchor than the positive (d(a,n)<d(a,p)). These are the triplets that will be penalized the most because if our objective is to bring the positives closer and separate the negatives this indicates the opposite.

- **Semi-hard triplets:** Triplets in which the distance from the negative to the anchor is not greater than that of the positive, but still has a loss greater than zero because it does not exceed the margin.

Ideally, easy triplets should be avoided for training since they give zero losses and the model takes longer to generalize. The figure 11 shows graphically in geometric space a representation of the types of triplets depending on the distance of their components from each other. "a" is the anchor, "p" is the positive, and the rest indicates which type of negative would be depending on its position.

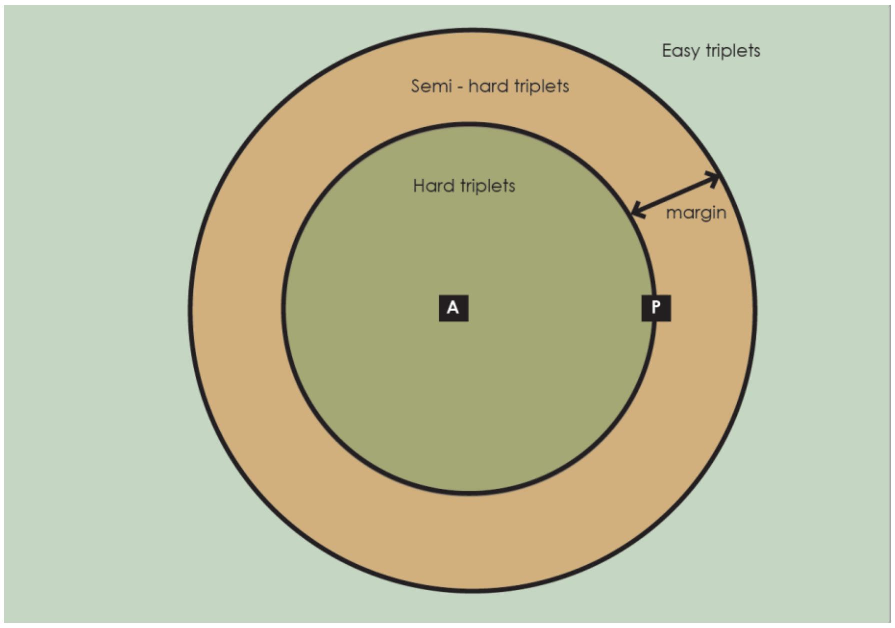


Figure 11. Visual representation of triplet types based on the negatives

5 MATERIAL AND METHODS

This section of the thesis gives a detailed account of the procedure that was followed in completing the project as well as the materials used in each step. To carry out this project we have followed a methodology divided into several phases listed below:

- **Preliminary investigation of the state-of-the-art:** We begin with an exploration of the various scientific articles and books that deal with the field of natural language processing.

- **Preliminary exploration of algorithms:** We explored the different implementations that people have made of the algorithms and loss functions that we were going to use for our project.

- **Web scraping:** To start working on the project, we created a database of documents with text information from the web on different diseases. Each disease is contained in one independent text file.

- **Building our model:** To build our model and choose the most suitable architecture to meet our objectives, we have made use of the knowledge gathered in the previous steps.

- **Training:** Once our model was built, we trained it by testing different settings of the model in order to compare different results.

- **Result exploration:** Finally, we explored the results by drawing conclusions about the performance of our model in the different circumstances in which we have tested it.

The following sections outline the steps that have been followed in these different phases of the project, detailing the materials and methods used for each one:

5.1 Preliminary investigation

5.1.1 State-of-the-art

Before starting to work on our project we have had to collect information in order to understand the state-of-the-art of all the branches of knowledge that our project touches. We have not only explored the current methods but also the precursors of these methods and their alternatives to opt for a deeper understanding of the subject.

For this we have resorted to books found to read in their electronic and physical format and above all to scientific articles by the authors of these methodologies and algorithms. All these documents we have found using Google Scholar, a search engine dedicated to scholarly literature across a wide variety of disciplines and sources: articles, theses, books…

On the other hand, the papers that explain algorithms and new proposals in the field of machine learning are usually of a mathematical nature and assume that the reader has the necessary scientific background to understand what they propose. To better understand the complex concepts found in the different papers, we have also had the help of blogs and didactic audio-visual products found on the web, which explain the subject in a more open and simplistic way.

5.1.2 Algorithms

Since we are dealing with mathematically complex topics, it is of great help to learn from the implementations that other people have made of algorithms and functions that we are interested in using. For this we have mostly resorted to code that can be found on GitHub. This is a platform that supports developers and allows them to collaborate and share their projects. It is mainly in remote teamwork as it allows development teams to work together on the same project and easily create new versions of software without disrupting the current versions.

These projects can be made public so that everyone can enjoy the progress made in this project. The implementations we found on GitHub are for completely different tasks than the ones we are interested in, but it has made things much easier to have a guide of architectures and structures that work using similar algorithms or based on the same idea.

5.2 Web Scraping

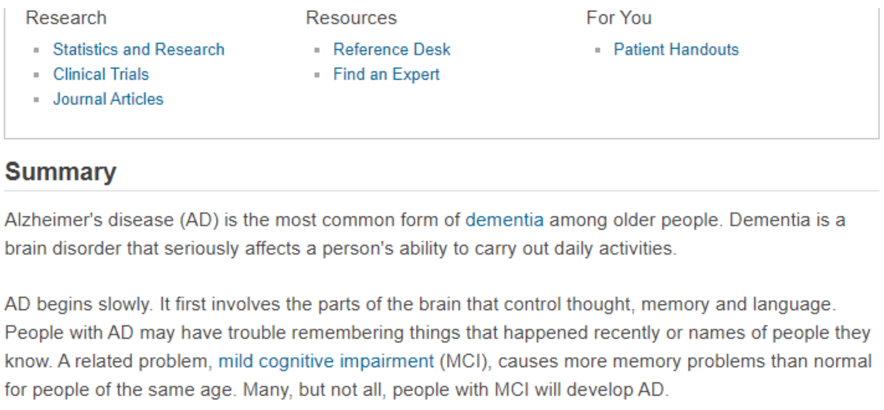
To create the database used in the training of the model we have obtained all the information we could using web-scraping methods. The information we have decided to collect comes from the MedlinePlus website, this is a quality and reliable medical information website that is open to everyone at all times. The fact that it is non-profit and non-advertising makes it easy to extract the information we need without too much noise. All content on the site is produced by the U.S. National Library of Medicine (NLM). NLM, the world's largest medical library, part of the U.S. National Institutes of Health (NIH).

Web scraping, or web data extraction is a method used for extracting data from websites [25]. This process can be carried out manually by extracting information from web pages, but the term web-scraping usually refers to the automation of processes that carry out the task of data extraction using bots or automated web browsers. Web pages are built using text-based mark-up languages (HTML and XHTML), and frequently contain a wealth of useful data in text form. This is why in recent years and given the increasing demand for data, new specialized tools have been developed for inspecting HTML code and extracting the information that these text tags contain.

The process for web scraping in this project can be broadly categorized into three steps:

- Understand and inspect the web page to find the HTML markers associated with the information we want.
- Use a specialized tool such as Selenium and/or other Python libraries to scrape the HTML page.
- Manipulate the scraped data to get it in the format we need.

The first step involves exploring the MedlinePlus webpage so we can understand how the data is distributed across the user interface. Once we have located what information we are interested its position we need to inspect the HTML markers that refer to our needed data. Each piece of text on a web page has its own html marker assigned to it that contains the information and is accessible to an automated bot. This is done because bots are nothing like human, you need to give him instructions to find your desired data since it cannot just look around the page until it sees something that matches our needs. HTML is the basic component of webs, It defines the meaning and structure of web content in the most organized and hierarchical way possible, making it relatively easy to automate search and text extraction processes according to our established parameters. In the figures 12 and 13 is represented the information about Alzheimer's disease in two different ways: as seen in the web page intended for users and as seen in HTML when inspecting the basic code of the web.

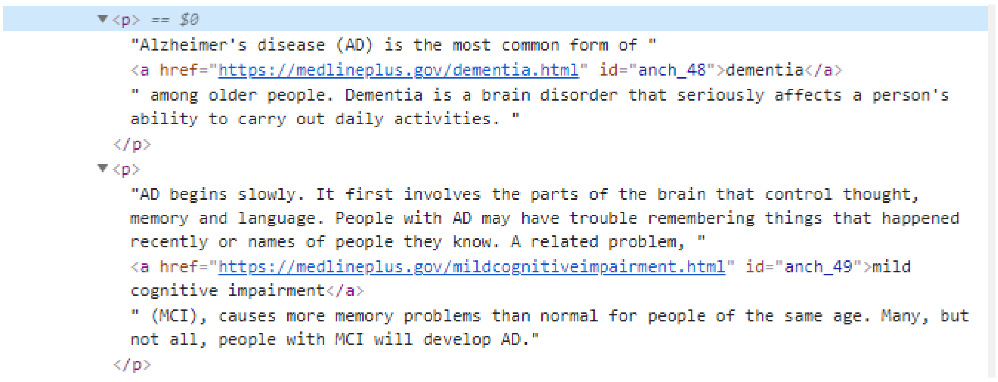


Summary

Alzheimer's disease (AD) is the most common form of dementia among older people. Dementia is a brain disorder that seriously affects a person's ability to carry out daily activities.

AD begins slowly. It first involves the parts of the brain that control thought, memory and language. People with AD may have trouble remembering things that happened recently or names of people they know. A related problem, mild cognitive impairment (MCI), causes more memory problems than normal for people of the same age. Many, but not all, people with MCI will develop AD.

Figure 12. Web page user interface



```
▼<p> == $0
   "Alzheimer's disease (AD) is the most common form of "
   <a href="https://medlineplus.gov/dementia.html" id="anch_48">dementia</a>
   " among older people. Dementia is a brain disorder that seriously affects a person's
   ability to carry out daily activities. "
 </p>
▼<p>
   "AD begins slowly. It first involves the parts of the brain that control thought,
   memory and language. People with AD may have trouble remembering things that happened
   recently or names of people they know. A related problem, "
   <a href="https://medlineplus.gov/mildcognitiveimpairment.html" id="anch_49">mild
   cognitive impairment</a>
   " (MCI), causes more memory problems than normal for people of the same age. Many, but
   not all, people with MCI will develop AD."
 </p>
```

Figure 13. Web page HTML

It can be seen that web page elements are represented by tags in HTML, for example, a paragraphs are writen between the tags <p> and </p>. These tags help our scraping tools identify each element making easier the search.

For the second step we used Selenium library for python. Selenium library allows you to automate web browsers. It is a powerful tool that allows us to use a web browser as if it were a human. With Selenium you can control any web browser by programming instructions using python code that are executed in the order you specify. The possibilities of these commands are almost unlimited in terms of what a human can do while browsing a web page: you can activate buttons, navigate from one page to another, enter text, passwords...

We have developed a script to enter the MedlinePlus page, navigate to the alphabetical list of topics in the form of hyperlinks, find the links to which these links point using HTML tags and save all the links to all the topics in a list. Using this list with URLs makes it easy for selenium to open one by one all the pages to extract information. For each topic, our script searches for the paragraphs that contain the central information and stores that text in a variable.

In order to manipulate the information and to be able to use it in the way we are interested in, we save the extracted data in separate text documents. Each text belonging to the same topic in a separate document. In a later step, we divide each text into sentences and pass them through the sentence embedding model we want to use, obtaining for each text as many vectors of the same size as sentences in the document. These vectors are saved in a dataframe and can be loaded quickly to train our model without the need to encode again all the sentences.

5.3 Building and training our model

The following sections explain the code attached in the appendix with which we have built the model. To program the tool, we have used the Python programming language. In addition to Python, we have used some libraries such as Keras, TensorFlow. matplotlib, pyPlot or Pandas [26][27][28][29] that facilitate the tasks that have been necessary to reach the established goals.

- **Keras** is a deep learning framework that makes easy the building of models and complex architectures. With Keras you can build layers one on top of each other and train the model with just a few lines. It uses TensorFlow for the maths.

- **TensorFlow** is an end-to-end open source platform for machine learning. It was developed by google and it is focused on the training of deep neural networks.

- **NumPy** is python library that provides tools which facilitate and optimize the mathematical operations needed for preprocessing of data. These functions are mostly focused in support for multi-dimensional arrays and matrices.

- **MatPlotLib** is a plotting library for creating static, animated and interactive visualizations in Python. We used it for data exploration.

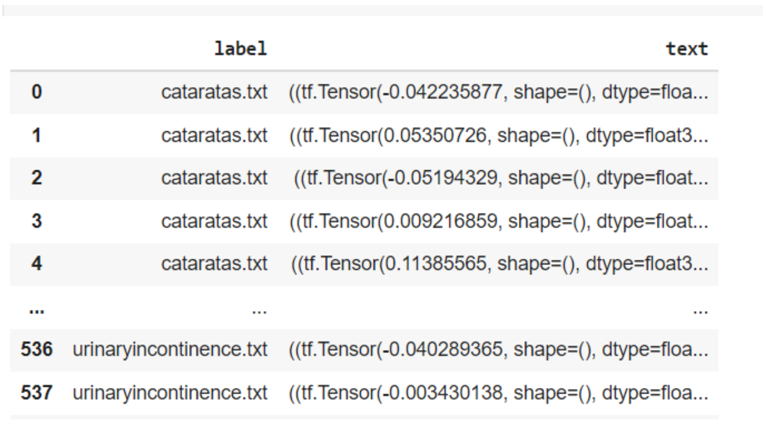
- **Pandas** is a Python library specialized in handling and analyzing data structures. We use ir to easily create dataframes

The code is structured in data preprocessing, build of used functions, arquitecture and training of the model, result exploration and prediction tool. Let's take a closer look at each of them.

5.3.1 Data preprocessing

Our data consists of all the documents that we have extracted from MedlinePlus. These documents have a file name that corresponds to the disease it deals with and contains the extracted text containing information about that disease. To create the data frame that we will use to train our model we have used pandas. The process consists of a loop that repeats an algorithm for each text: It opens the file, divides the text into lines and separates them into sentences when they are divided by dots. Each of the sentences is processed by the Universal Sentence Encoder (USE) and the encoded output is stored in a list. We have used this model because our data is in a language other than English and USE has a multilingual version for foreign languages.

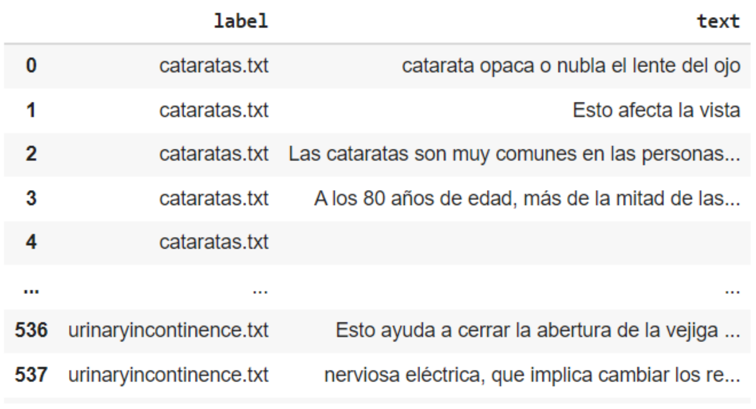
In parallel, labels of each sentence are stored in another list. Finally, when all sentences of all texts have been processed, a dataframe is created with two columns: "label" and "text". In the "labels" column are the classes to which each of the phrases they refer to belong, which are stored in the "text" column as vectors.



	label	text
0	cataratas.txt	((tf.Tensor(-0.042235877, shape=(), dtype=floa...
1	cataratas.txt	((tf.Tensor(0.05350726, shape=(), dtype=float3...
2	cataratas.txt	((tf.Tensor(-0.05194329, shape=(), dtype=float...
3	cataratas.txt	((tf.Tensor(0.009216859, shape=(), dtype=float...
4	cataratas.txt	((tf.Tensor(0.11385565, shape=(), dtype=float3...
...	...	...
536	urinaryincontinence.txt	((tf.Tensor(-0.040289365, shape=(), dtype=floa...
537	urinaryincontinence.txt	((tf.Tensor(-0.003430138, shape=(), dtype=floa...

Figure 14. Embedding data frame

In order to explore the data after training, a dataframe is created identical to the previous one but without being processed by the USE. This way we will have the real text that we can read and understand easily.



	label	text
0	cataratas.txt	catarata opaca o nubla el lente del ojo
1	cataratas.txt	Esto afecta la vista
2	cataratas.txt	Las cataratas son muy comunes en las personas...
3	cataratas.txt	A los 80 años de edad, más de la mitad de las...
4	cataratas.txt	
...	...	...
536	urinaryincontinence.txt	Esto ayuda a cerrar la abertura de la vejiga ...
537	urinaryincontinence.txt	nerviosa eléctrica, que implica cambiar los re...

Figure 15. Translation data frame

In a next step, using this data frame, all sentences consisting of an empty array are removed from the dataset, in the figure 15 you can see that 4th sentence does not contain text so it will be removed. Once the data is cleaned, it is randomly divided and separated into 80% for training and 20% for validation and the labels are stored in a variable 'Y' and the embeddings in a variable 'X', leaving the data ready to be fed to the model..

5.3.2 Functions

The two functions that had to be developed for the project are the data generator and the triplet loss cost function. These functions are used by the model during training. The data generator feeds the model with three-by-three encodings (anchor, positive and negative). The triplet loss function calculates the cost of the model output, this numerical value will be minimized by modifying the weights of the neural network.

The data generator accepts the variables X and Y with the data and generates the triplets as outputs. For this it selects a random phrase from the X list to use as anchor, and searches for its label in the Y list. Then it selects a phrase from the same class and another from a different class. These will be the positive and negative respectively. This process is repeated as many times as the batch size (bs) value, generating bs triplets. Finally, it saves all the triplets in an array and this batch is outputed to feed the network. Being a distance minimization learning, the Y labels are only used to find the triplets and are not returned. This is the simplest way to find triplets, the problem is that it is not optimal since many times the triplets will be easy triplets, which will have a loss of 0, so it will not provide information to the model. Ideally only semi-hard and hard triplets should be selected.

The loss function has as input only the margin and the encodings that come out of the network, the cost of the model is calculated with these three results so it is not necessary to compare it with the input data of the model. These encodings come out of the model as a concatenated vector, so the first thing the function will do is to divide this vector into three equal parts, obtaining the encoding of the anchor, the positive, and the negative. As a second step it calculates the distance from the anchor to the positive and the negative using squared Euclidean distance. This distance d is defined by:

$$d(a,p) = (f(x_a) - f(x_p))^2 \qquad (3)$$

$$d(a,n) = (f(x_a) - f(x_n))^2 \qquad (4)$$

Where a, p and n are anchor, positive and negative respectively, and f(x) are the encodings coming out of the model.

Finally, the loss is calculated, which is given by the Equation 2.

$$\mathcal{L} = \max(d(a,p) - d(a,n) + \alpha, 0) \qquad (2)$$

Where alpha is the margin that is forced between positive and negative. This loss is what the model tries to minimize in each iteration.

5.3.3 Model

The architecture used for the model is quite simple, consists of three inputs that connect to a dense layer common to the three inputs, a lambda layer for normalization and a layer that concatenates the three encodings into one vector.

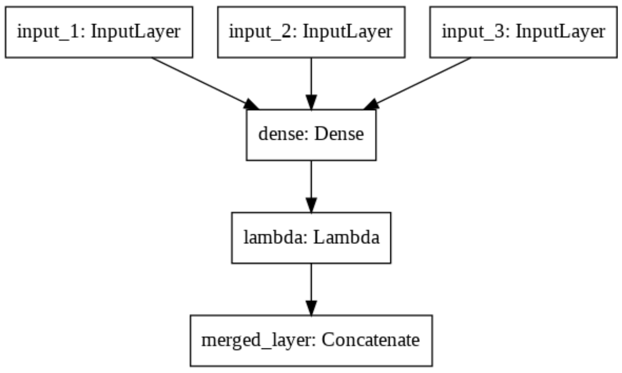
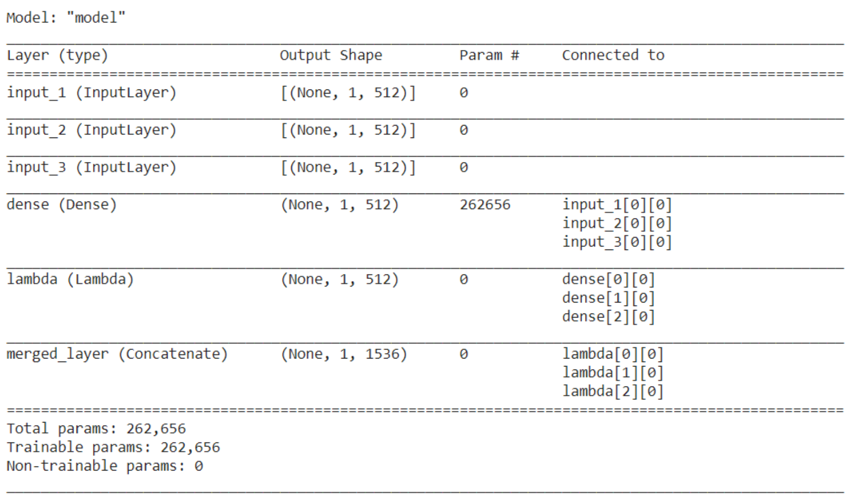


Figure 16. Architecture of our approach

The inputs are three encodings (anchor, positive and negative) of dimensionality 512, since we have used the Universal Sentence Encoder to process the sentences, which generates embeddings of 512 floating points. These inputs are processed in the dense layer, which is the only part of the model with trainable parameters. For this layer we have used sigmoid activation as it gave much better results than the other activation functions. The embeddings coming out of this layer are normalized by L2 normalization and finally concatenated into a single vector that will be used to calculate the cost. By concatenation, the output of the model will be of dimension 1536 (512*3). The figure 17 shows the summary of this model.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1, 512)]     0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1, 512)]     0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1, 512)]     0
__________________________________________________________________________________________________
dense (Dense)                   (None, 1, 512)       262656      input_1[0][0]
                                                                 input_2[0][0]
                                                                 input_3[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 1, 512)       0           dense[0][0]
                                                                 dense[1][0]
                                                                 dense[2][0]
__________________________________________________________________________________________________
merged_layer (Concatenate)      (None, 1, 1536)      0           lambda[0][0]
                                                                 lambda[1][0]
                                                                 lambda[2][0]
==================================================================================================
Total params: 262,656
Trainable params: 262,656
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 17. Detailed architecture

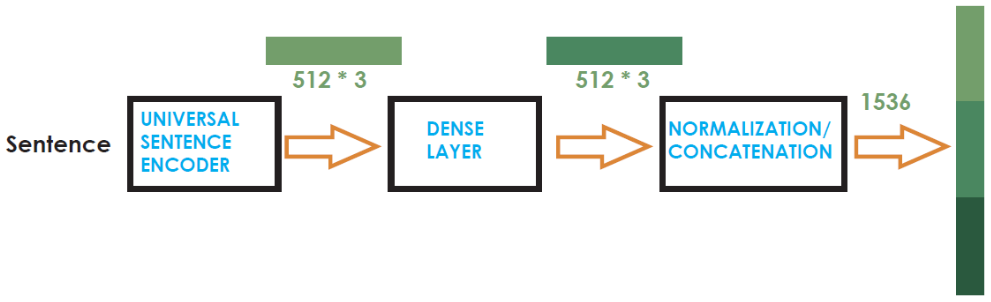


Figure 18. Generic architecture

Seen globally, the encoding process of a sentence would be to first use a pre-trained sentence encoder to take advantage of its generic knowledge of the language, the resulting embedig to introduce it in our model and obtain its final encoding (figure 18).

Finally we have trained the model with a batch size of 16 and 100 steps per epoch until it covered. In order to optimize training times we have used the Google Colab platform that allows using GPU to run Python notebooks. Figure 19 shows the training and validation graph.

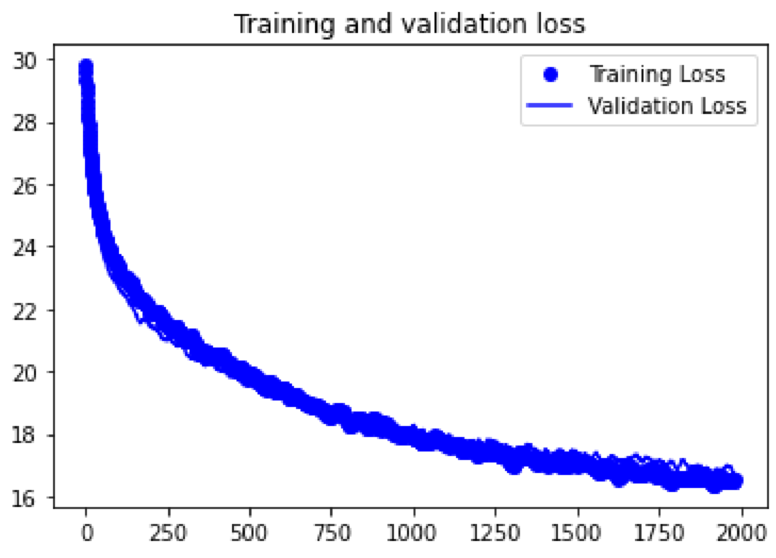


Figure 19. Training los graph

5.4 Result Exploration

For the exploration of the results, it has been necessary to first process the set of embeddings that we have obtained, since being high dimensionality vectors it is not possible to interpret the results by means of graphs. For this we have used the t-SNE algorithm that reduces the dimensionality of the data. Once processed, it has been possible to use Python visualization tools to interpret the geometric space.

To obtain quantitative results, a function has been constructed that attempts to classify the test set sentences using Euclidean distance between points. This function calculates a hit percentage based on whether the actual class appears among the predictions made.

5.4.1 T-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a data exploration technique developed by Laurens van der Maatens and Geoffrey Hinton in 2008 for dimensionality reduction. It is an unsupervised, non-linear method that stands out for visualization of high-dimensional data. In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high-dimensional space [30].

In the case of our project, it is very convenient for us to visualize our data in a lower dimension than the original one. Since we work with encodings of sentences, with many features for each of the data, we end up with an algebraic space of too many dimensions (512 or 1024) to be represented and understood by a human. T-SNE is especially useful in segmentation problems where you need to see if clusters are forming in your data. This information cannot be obtained from a geometric space with hundreds of dimensions, in order to understand how the data is distributed it is necessary to represent it with a dimensionality that humans can understand: 1-D 2-D or 3D.

Let’s see an example: In the figure 20 below, you can see some data in clusters in the 2-dimensional space. If we want to lower its dimension to 1-D, we could simply project the data onto just one of its dimensions, but this will give as a result the overlap of at least two of the clusters.

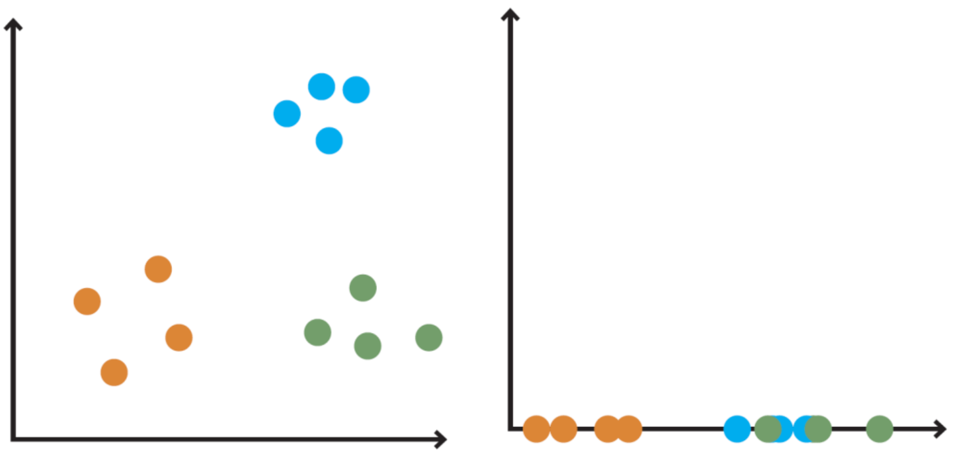


Figure 20. projection of points to X-axis

This is not a good representation of our data, since what we are interested in from the visualization is to know if they are grouped or if the classes are close or far from each other in the vector space. The 1-D projection implies that the data of classes 1 and 2 are mixed, but as we see in the original data distribution, they are separated and grouped together. A good representation of this 1-D data would be the figure 21, in which, despite the reduced dimensionality, the samples maintain a similar distribution.

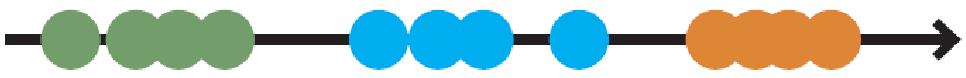


Figure 21. Correct representation of points in 1-D

This representation is what t-sne is all about, that no information is lost by reducing the dimensionality of the data, so that accurate conclusions can be drawn simply by looking at it.

The algorithm first calculates a similarity measure between pairs of points in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function. Let’s break that down into 3 basic stages:

- **First stage:** First the algorithm calculates the Euclidian distances between pairs of points. To achieve this, it centers a gaussian distribution on the first point (x_i) and measure the distance to the rest of the poins (x_j). Then normalized for all points because it is not the density of the clusters that is relevant, but the fact that they are clusters. This transform the distances into conditional probabilties (Pij) representing the similarity between every two points. If two points are similar under this gaussian circle it means that they are likely to be neighbors in the hig-dimensional space.

- **Second stage:** This step is similar to the first one but in the target dimensionality (low-dimensional space) and using Student t-distribution instead of gaussian. To do that t-SNE builds a random dataset with the same number of instances as there are in the high dimensionality dataset. These points are generated with the same number of features as dimensions to which we want to reduce (2 features if we want to map our data to 2-D). It will be something like the figure 22. For this set of points we will calculate the same joint probability (Qij) but this time using Student t-distribution instead of Gaussian (hence the name t-SNE). Using this distribution avoids that the points in the low-dimensional space are too close together and cannot be distinguished.

Figure 22. Random distribution of points

- **Third stage:** In the last step t-SNE tries to make the joint probability distribution Qij of the data points in the low dimension as close as possible to the probabilities Pij of the high-dimensional space. As a loss function to calculate the difference between these two sets of probabilities the algorithm uses the Kullback-Lieber (KL) divergence [31]. Finally it uses gradient descent to minimize this KL loss function. From this optimization, we get the values of the points in the low dimension dataset and use it for our visualization.

5.4.2 Scatter Plot

To represent this data we have used the MatplotLib library for python, which offers a clean and easy to interpret representation.

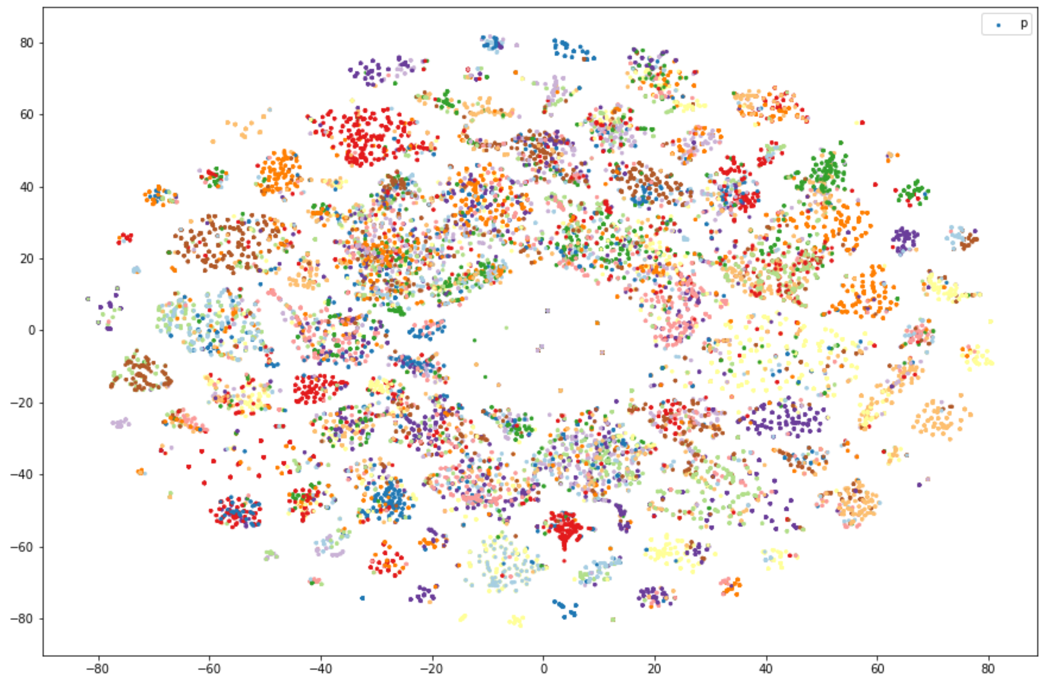


Figure 23. MatplotLib scatter plot

To go deeper into these graphs we have used the PlotLy library, which has allowed us to explore each of the points to interpret the results in more detail as it gives information about each point (label, text, etc...).

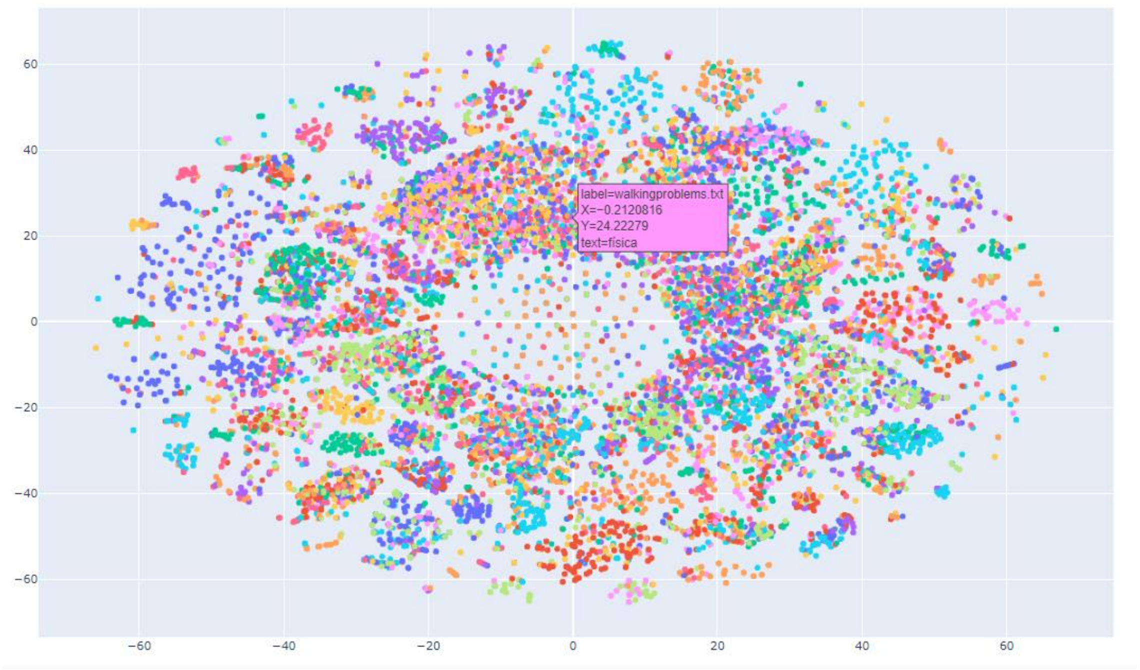


Figure 24. Detailed scatter plot

5.4.3 Prediction function

Finally, in order to test the model in an alternative way to the visual one, we have written a simple prediction function that allows us to classify sentences. This classification is done in the simplest possible way: we compute the embedding of the sentence we want to represent and position it in the vector space. To know its class we look for the nearest neighboring points and assign the class they have.

It should be noted that the objective of this model is not to build a perfect classifier, since there are already too many classes (the model was trained sentences from more than 1000 documents) very similar to each other and the data are sentences that could belong to many of them. What we are trying to do with this vectorization is to classify a phrase into a generic class or topic. For example if we introduce the phrase "among the symptoms is the intense pain in the left side of the chest", and the classes of the neighboring points that we obtain as a result are, "heart attack", "angina" and "pericarditis", we would take it as a success although the real class that we want to obtain is "heart failure". This is because it has succeeded in placing the input phrase in the zone of the embedding space that corresponds to heart diseases.

For this reason, to measure the accuracy we have created a function that has as input the phrase to classify, its real label and a t value (threshold). It returns the classes of the t nearest points, depending on the threshold that is introduced, being by default this value 5. It also returns a boolean that is True if the real class is among the classes of the neighbors, and False if not, this way, with the threshold it is possible to give a margin of error to the classifier and compute numerical values that reflect how close the model is to predicting the real classes of the sentences. With this classifier we can compute predictions for the test set data set and get a higher percentage of success which will be higher the higher the t-threshold.

6 RESULTS

The results obtained have been measured in two different ways, by a visual exploration and by using a prediction function in the test set that allows computing an accuracy value that gives an insight of how the model is performing.

6.1.1 Visual results

To have a first perception of how the model is working we have represented the embedding space in a 2- and 3-dimensional plane, in this way we can easily identify if clusters are forming between the vector representations of phrases that belong to the same class. In the figure 25 you can see this scatter plot in which each point is a phrase and each colour represents the class to which the point belongs

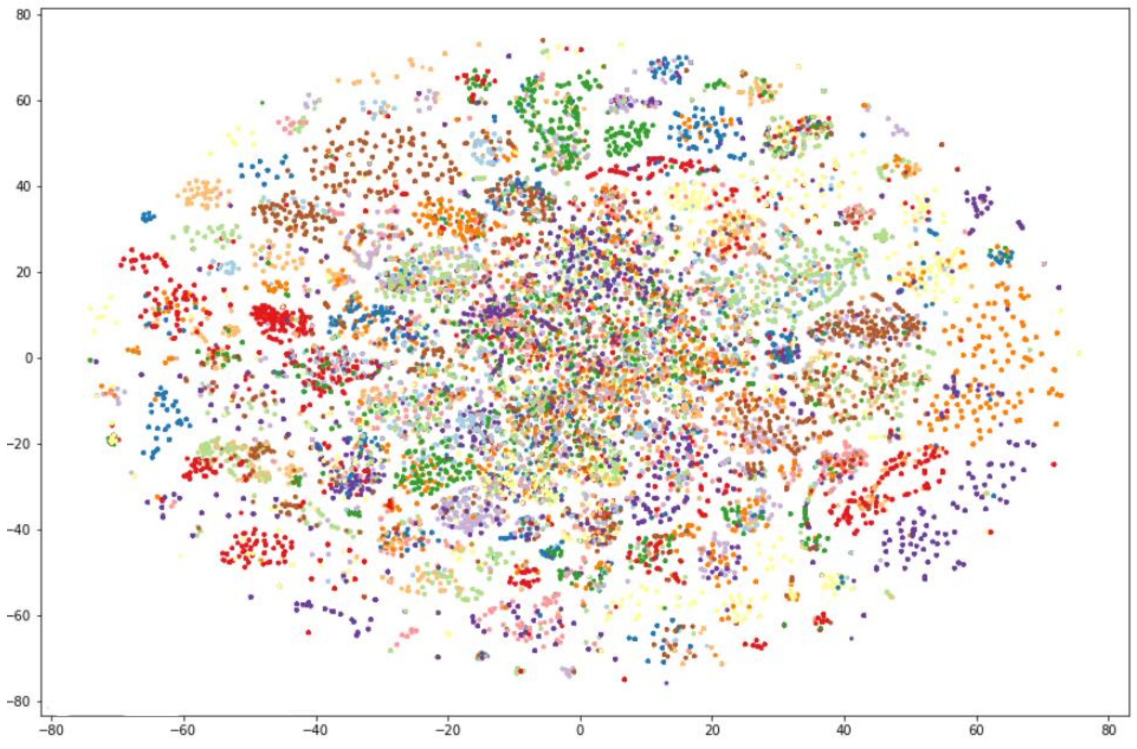


Figure 25. Embedding space in 2-D

We can also visualize de data un 3D, but it is harder to get an insight from this representation:

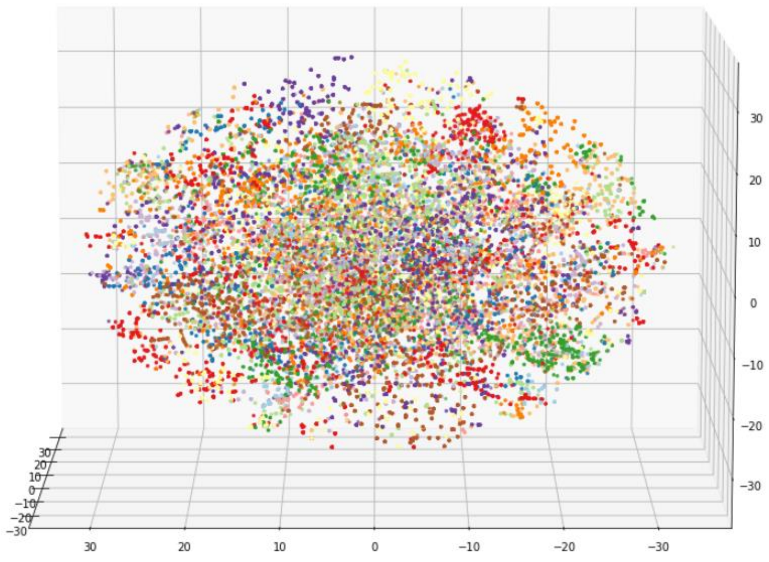


Figure 26. Embedding space in 3-D

Although this visualization gives us an idea of how the data is distributed, we cannot know to which class each cluster belongs and why it is being distributed in the geometric space. This is why we have also created an interactive scatterplot that allows us to analyze each cluster and even each class. When you hover the mouse over a point, it tells you what phrase it is and what class it belongs to. For example, let's see in detail what the accumulation of red dots of the coordinates (-50,-40) in the figure 25 is about.

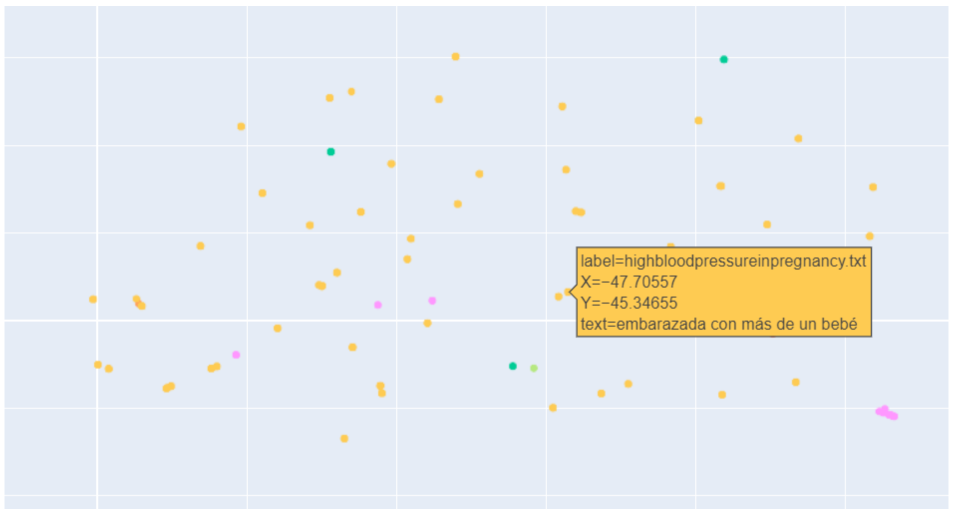

label=highbloodpressureinpregnancy.txt
X=−47.70557
Y=−45.34655
text=embarazada con más de un bebé

Figure 27. Cluster of points with same label

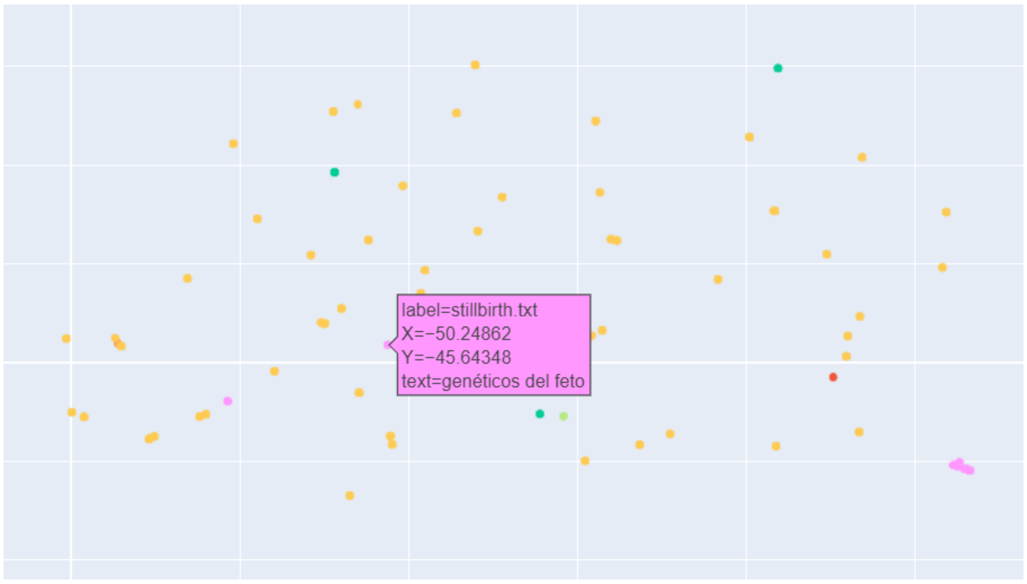
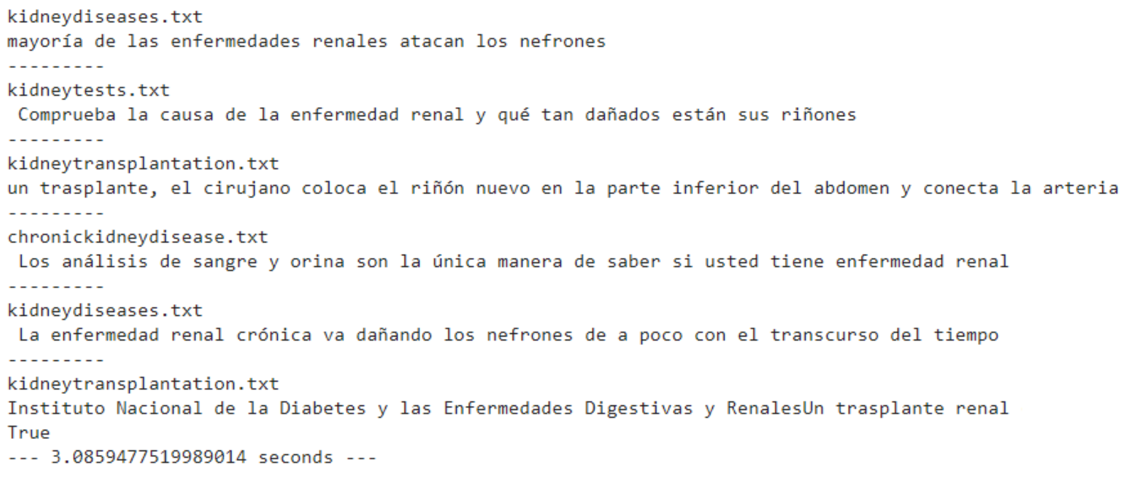


Figure 28. Point with different label in a cluster

As we can see in figures 27 and 28 most of them belong to the class "high pressure in pregnancy" and the phrases refer to pregnancy or related. The items that do not belong to this class refer to the same topics.

6.1.2 Quantitative results

We can use the prediction function in two ways, to classify single sentences or to calculate a model accuracy value. Let's see an example of the classification of an isolated sentence.



```
kidneydiseases.txt
mayoría de las enfermedades renales atacan los nefrones
---------
kidneytests.txt
 Comprueba la causa de la enfermedad renal y qué tan dañados están sus riñones
---------
kidneytransplantation.txt
un trasplante, el cirujano coloca el riñón nuevo en la parte inferior del abdomen y conecta la arteria
---------
chronickidneydisease.txt
 Los análisis de sangre y orina son la única manera de saber si usted tiene enfermedad renal
---------
kidneydiseases.txt
 La enfermedad renal crónica va dañando los nefrones de a poco con el transcurso del tiempo
---------
kidneytransplantation.txt
Instituto Nacional de la Diabetes y las Enfermedades Digestivas y RenalesUn trasplante renal
True
--- 3.0859477519989014 seconds ---
```

Figure 29. Result of classification function

The figure 29 shows the classification results. The first section indicates the sentence to be classified and its actual label. The following sections indicate the class of the nearest neighbours to this point. In this case we have used a threshold of 5, so the 5 nearest neighbours will be shown.

In these results it can be seen that the actual prediction was not correct until the third attempt, but all the attempts talk about the kidneys. This is the reason for including a threshold in the function, to give a margin since we are not interested in getting it right on the first try, but to place it in an area where the surrounding sentences deal with the same topic.

On the other hand, to obtain numerical values of the model performance, we iterate the test set with the function, and we take as a success the presence of the real class in the results that are returned. In this way, the higher the threshold, the higher the probability of success because more possible classes are returned. Once all the data has been iterated, the success percentage can be calculated.

Table 1. Quantitative results: Accuracy of the model

N	1	2	3	4	5	7	10	15	20
%	33.8%	42.1%	48.7%	54.6%	58%	62.8%	66.2%	69.6%	72.4%

7 DISCUSSION

In the scatter plots it can be seen that indeed the sentences belonging to the same class have been put together in the geometrical space. In the following figures 30 and 31 you can see a comparison of the plane with the data represented before being processed by our model and afterwards.

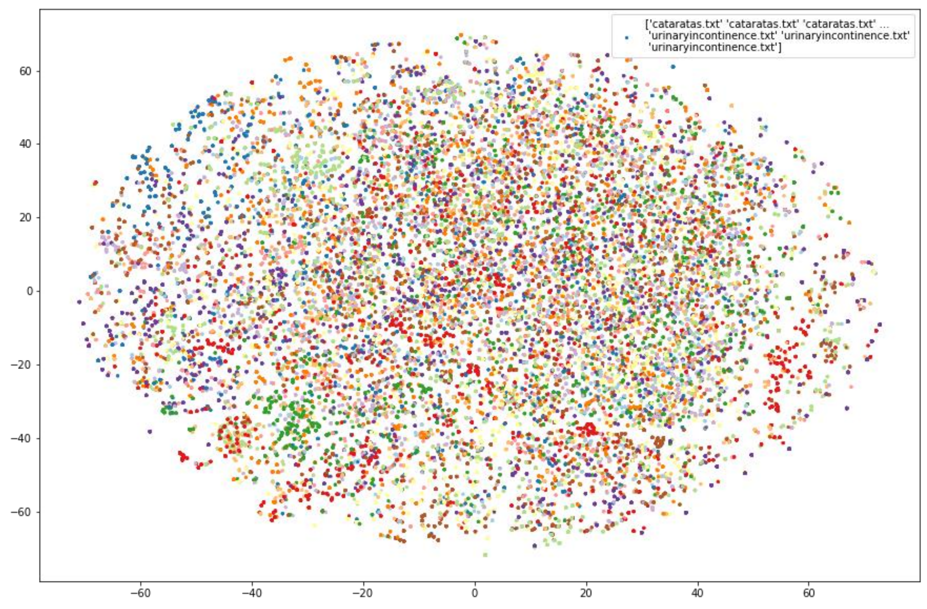


Figure 30. Embedding space before our embedding

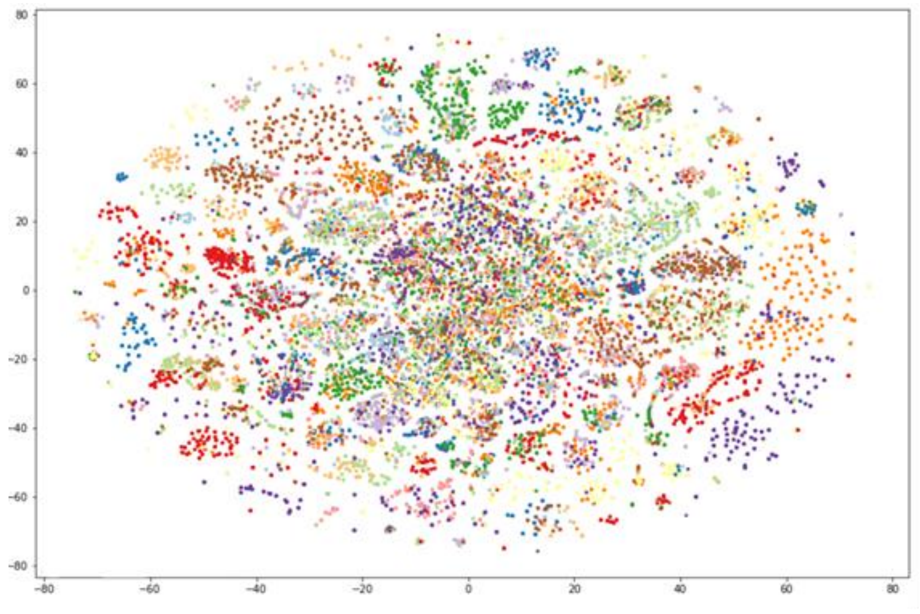


Figure 31. Embedding space after our embedding

It can be clearly seen how, despite the noise introduced by the data, many sentences have formed clusters when passing through the model. In the figure 32 you can see the representation of the data when the project started, using far fewer classes than we currently use to train the model. Generally, the clusters are well defined and separated from each other. This tells us that the triplet loss function is very susceptible to noise, since, when using an internet phrase database, there is a lot of data that is repeated or does not provide information about the class to which it belongs.

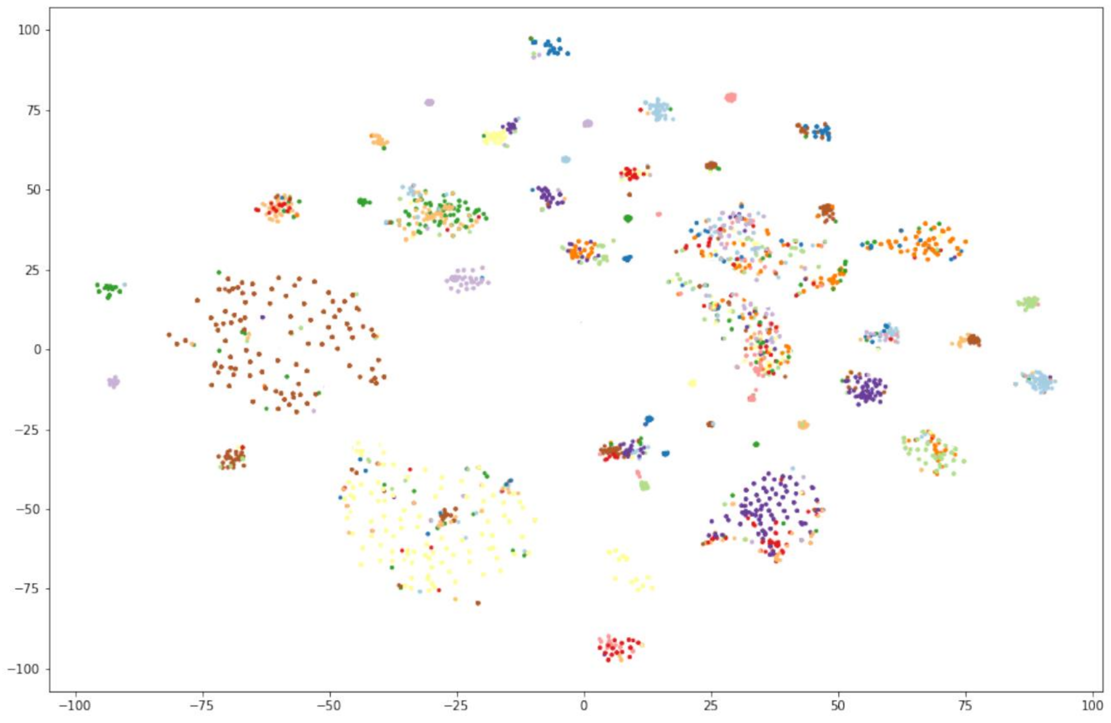


Figure 32. Early scatter plot of the embedding space

The points that are mixed at coordinates (30,25) are generic phrases that the model cannot classify as they do not provide information and are repeated for different classes. An example of this could be the sentence "Treatment can improve symptoms" or "In severe cases hospitalization is required". This type of sentence can talk about any disease that has treatment or is severe, which is why the model is not able to group them with their classes and they are considered noise for the model.

When exploring a particular cluster, it can be seen that although the classes do not coincide between the points, they deal with the same topic. In the example of the figure 33, the data are from different classes but all of them are about hair.

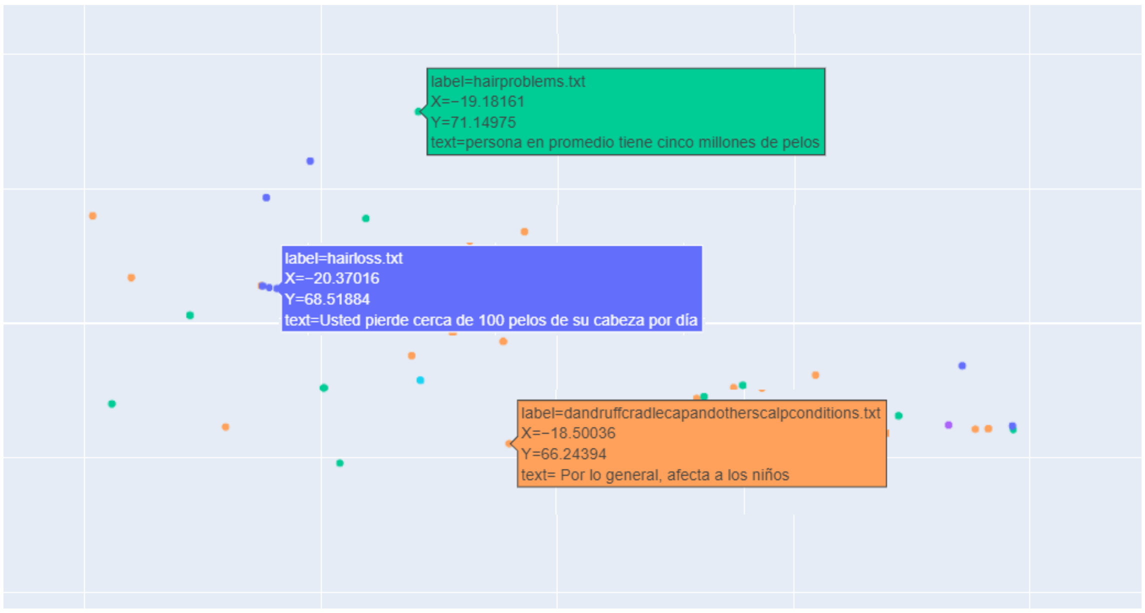


Figure 33. Cluster of noise points

In the same way, when trying to predict a point, the prediction function does not always get it right on the first try, but the results it displays are generally related to each other and to the sentence to be classified. In the example given in the "Results" section, the function does not get it right until the third attempt, but all the results are related to each other because they refer to diseases affecting the kidneys.

Finally, the accuracy table gives good results, a 33.8% hit rate with a single attempt, although not a good score for a classifier, indicates that the model is really bringing embeddings of the same class closer together. As the threshold is increased, the accuracy increases dramatically to 62.8% when it reaches 7 attempts. Setting the threshold higher than this value does not make much difference. This may be because, if the class is not among the 7 closest points, it is most likely a misclassified phrase.

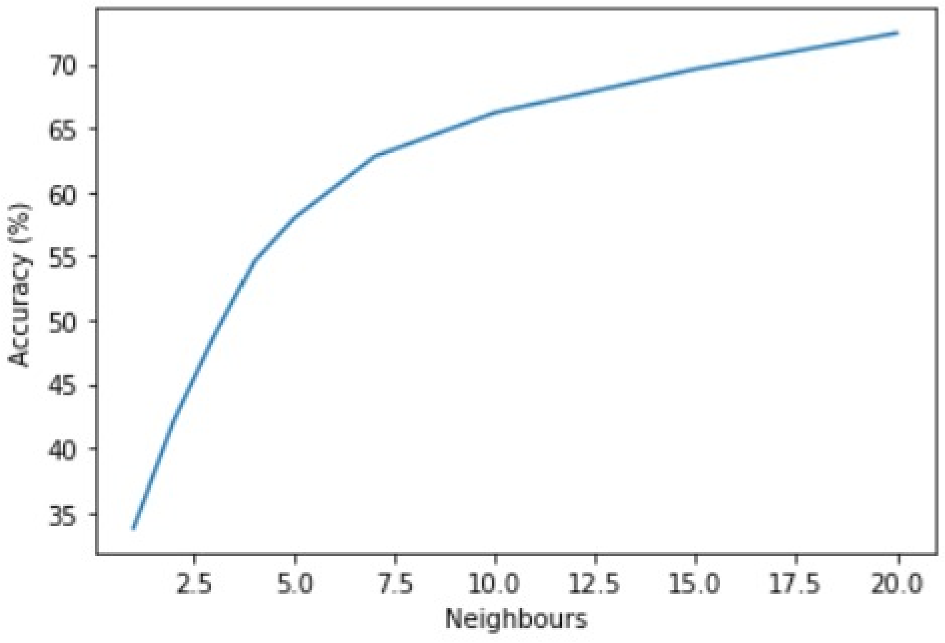


Figure 34. Accuracy Graph

The graph shows how the accuracy converges as the threshold of the prediction function increases.

With this combination of techniques to explore the data we can conclude that the result has been satisfactory. The aim of this project was to explore the possibilities presented by new advances in artificial intelligence and to put them into practice in order to improve, as far as possible, the state of the art of NLP. We have used a loss function that until now had been used almost exclusively for image processing and we have obtained results that show that it also improves results for specific NLP tasks that pre-trained models cannot cover.

8 CONCLUSIONS

According to the objectives that had been set, which were to achieve a model capable of representing sentences in such a way that they are grouped in the geometric space according to their biomedical relationships, the results have been satisfactory. Depending on the threshold we set, good results can be obtained, indicating that the model is capable of grouping the sentences according to their medical subject. So it can be concluded that it is feasible to use distance learning such as triplet loss to improve NLP models performance for specific tasks.

As future lines we could consider the comparison of performances using different pre-trained models such as BERT. We could also improve the selection of triplets to accelerate training and with more computational resources it would be possible to increase the batch size for generalization. It would also greatly improve the quality of the model to increase the quality of the data by reducing the noise in it.

9 REFERENCES

[1] Lopez Pizarro, D. “Feeding robotic arm prototype based on mouth spatial positioning”. Ceu San Pablo, 2021

[2] Stevenson, M., Guo, Y., Gaizauskas, R., & Martinez, D. (2008). Disambiguation of biomedical text using diverse sources of information. BMC bioinformatics, 9(11), 1-11.

[3] Chen, Q., Peng, Y., & Lu, Z. (2019, June). BioSentVec: creating sentence embeddings for biomedical texts. In 2019 IEEE International Conference on Healthcare Informatics (ICHI)

[4] Deng, L., Li, J., Huang, J. T., Yao, K., Yu, D., Seide, F., ... & Acero, A. (2013, May). Recent advances in deep learning for speech research at Microsoft. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing

[5] Ogiela, M. R., & Ogiela, U. (2010). The use of mathematical linguistic methods in creating secret sharing threshold algorithms. Computers & Mathematics with Applications

[6] Liddy, E. D. (2001). Natural language processing.

[7] Guo, D., Zhou, W., Li, H., & Wang, M. (2017). Online early-late fusion based on adaptive HMM for sign language recognition. ACM Transactions on Multimedia Computing, Communications, and Applications

[8] Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. The journal of machine learning research, 3, 1137-1155.

[9] “Introduction to Word Embeddings and its Applications” https://medium.com/compassred-data-blog/introduction-to-word-embeddings-and-its-applications-8749fd1eb232

[10] “A simple Word2vec tutorial” https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

[11] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems.

[12] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing

[13] “Intuitive Guide to Understanding GloVe Embeddings” https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[14] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365.

[15] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[16] “What are Sentence Embeddings and why are they useful?” https://engineering.talkdesk.com/what-are-sentence-embeddings-and-why-are-they-useful-53ed370b3f35

[17] A gentle introduction to Doc2Vec

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems

[19] “Neural Machine Translation with Transformers” https://galhever.medium.com/neural-machine-translation-with-transformers-69d4bf918299

[20] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

[21] “The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)” http://jalammar.github.io/illustrated-bert/

[22] Cer, D., Yang, Y., Kong, S. Y., Hua, N., Limtiaco, N., John, R. S., ... & Kurzweil, R. (2018). Universal sentence encoder

[23] Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

[24] Koch, G., Zemel, R., & Salakhutdinov, R. (2015, July). Siamese neural networks for one-shot image recognition. In ICML deep learning workshop (Vol. 2).

[25] “Web scraping” https://en.wikipedia.org/wiki/Web_scraping

[26] https://keras.io

[27] https://www.tensorflow.org/?hl=es-419

[28] https://matplotlib.org

[29] https://pandas.pydata.org/

[30] “Introduction to Word Embeddings and its Applications” https://medium.com/compassred-data-blog/introduction-to-word-embeddings-and-its-applications-8749fd1eb232

[31] Belov, D. I., & Armstrong, R. D. (2011). Distributions of the Kullback–Leibler divergence with applications. British Journal of Mathematical and Statistical Psychology